

AD-A255 945



Task/Subtask IS15.02
CDRL Sequence 03705-001
30 September 1991

(2)

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Software Process Tools and Techniques Evaluation Report

Version 1.0

Contract No. F19628-88-D-0032

Task IS15-Software Process Management

DTIC
ELECTE
SEP 29 1992
S A D

Prepared for:

**Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000**

Prepared by:

**IBM Federal Sector Division
800 North Frederick Avenue
Gaithersburg, MD 20879**

This document has been approved
for public release and sale, its
distribution is unlimited.

92 9 28 107

422287

92-26078



173P8

This document has been approved for public release.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0180

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204 Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0180), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank) 2. REPORT DATE 3. REPORT TYPE AND DATES COVERED

September 30, 1991 Initial

4. TITLE AND SUBTITLE

Software Process Tools and Techniques Evaluation Report

5. FUNDING NUMBERS

F19628-88-D-0032/0005

6. AUTHOR(S)

William H. Ett, IBM

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

IBM Federal Sector Division
800 North Frederick Avenue
Gaithersburg, MD 20879

8. PERFORMING ORGANIZATION
REPORT NUMBER

03705-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Electronic Systems Division/AVK
Air Force Systems Command, USAF
Hanscom Air Force Base, MA 01731-5000

10. SPONSORING/MONITORING
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Cleared for public release

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This document describes the tools and technology examined on STARS Task IS15. This report summarizes the IBM team's examination of software process representation tools and techniques. It also summarizes the examination of software process enactment tools, and techniques for implementing a process system from a well-defined system of processes, such as the "Cleanroom Engineering Software Development Process".

The software process definition tools and techniques sections of the document: 1) examines the feasibility of porting the Software Process Management System (SPMS) from the Apple Macintosh to the IBM STARS SEE, 2) provides a SPMS Port plan and 3) discusses the use of box structures as a notation for recording aspects of software processes. The software process enactment tools and techniques sections of the document: 1) describes the KI Shell tool selected for supporting the IBM STARS "Cleanroom Software Process Case Study", 2) describes the specification, design and implementation of the "Cleanroom Engineering Process Assistant" prototype, and 3) provides lessons learned from performing the "Cleanroom Software Process Case Study". Finally the document makes recommendations for the selection of software process definition and enactment support capabilities for the IBM STARS SEE.

14. SUBJECT TERMS KI Shell, Case Study, Software Process Enactment, Cleanroom Engineering, Software Development Process, Software Process Management System, Process Definition Tools, Process Representation Techniques, Software Process Enactment Tools

15. NUMBER OF PAGES

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION
OF REPORT

Unclassified

18. SECURITY CLASSIFICATION
OF THIS PAGE

Unclassified

19. SECURITY CLASSIFICATION
OF ABSTRACT

Unclassified

20. LIMITATION OF ABSTRACT

SAR

This document was developed by the IBM Federal Sector Division, located at 800 North Frederick Avenue, Gaithersburg, MD 20879. Questions or comments should be directed to the document owner and author, William H. Ett, IBM, (Internet: ETTB@WMAVM7.iinus1.ibm.com).

This document is approved for release under Distribution "C" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24). Permission to use, modify, copy or comment on this document for purposes stated under Distribution "C" without fee is hereby granted. The Government (IBM and its subcontractors) disclaims all responsibility against liability, including expenses for violation of proprietary rights, or copyrights arising out use of this document. In addition, the Government (IBM and its subcontractors) disclaims all warranties with regard to this document. In no event shall the Government (IBM nor its subcontractors) be liable for any damages in connection with the use of this document.

Accession For	
NTIS	CRAM
DTIC	FILE
Unannounced	
Justification	
By	
Distribution	
Availability	
Doc	
A-11	

ITEM QUALITY INSPECTED 3

STARS Task IS-15
Software Process Tools and Techniques Evaluation Report
Version 1.0

23 September 1991

W. H. Ett, IBM
R. H. Cobb, SET
Herb Krasner, SAIC
Ara Kouchakdjian, SET
Susan Phillips, SAIC
Jay Ramanathan, UES
Rajiv Ramnath, UES
Bruce Reed, UES
Jim Terrel, SAIC

International Business Machines Corporation
Federal Sector Division
System Environments
800 North Frederick Avenue
Gaithersburg, Maryland 20879

Intentionally left blank.

Preface

Preface

This report describes the activities and lessons learned during the performance of IBM's "S" increment process tasks. The IBM STARS "S" Increment Process Task Team was composed of personnel from:

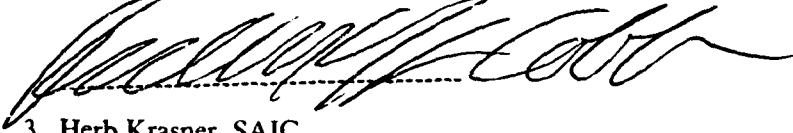
- IBM Federal Sector Division, Software Technology and Products, Gaithersburg, Maryland
 - William H. Ett, IBM
- Science Applications International Corporation (SAIC), Software Technology Center, Austin, Texas
 - Herb Krasner, SAIC
 - Susan Phillips, SAIC
 - Jim Terrel, SAIC
 - Adam Linehan, SAIC
- Software Engineering Technology, Incorporated (SET), Annapolis, Maryland
 - Richard H. Cobb, SET
 - Ara Kouchakdjian, SET
 - Roger Sisson, SET
- UES, Incorporated, Columbus, Ohio
 - Jay Ramanathan, UES
 - Rajiv Ramnath, UES
 - Bruce Reed, UES
 - Venhat Ashok, UES.

Author Sign-Off

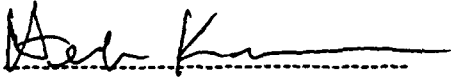
1. W. H. Ett, IBM



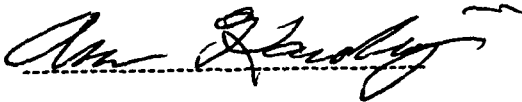
2. R. H. Cobb, SET



3. Herb Krasner, SAIC



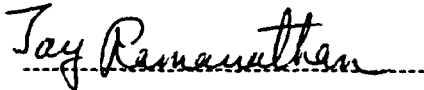
4. Ara Kouchakdjian, SET



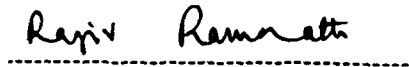
5. Susan Phillips, SAIC



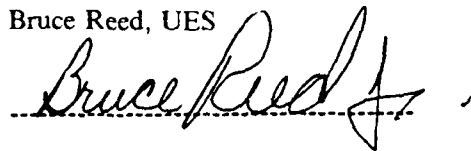
6. Jay Ramanathan, UES



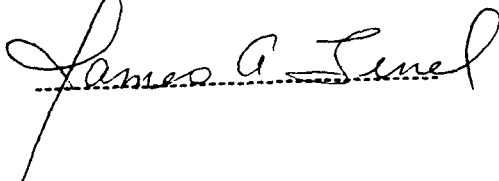
7. Rajiv Ramnath, UES



8. Bruce Reed, UES



9. Jim Terrel, SAIC



Contents

1.0 Document Introduction	3
1.1 Audience of Document	3
1.2 How this Document Can Be Used	3
1.2.1 Background	3
1.2.2 Provide Reader with Basic Concepts of SPMS	4
1.2.3 Provide Reader with a Case Study of Implementing a Defined Process	4
2.0 IBM STARS Task IS-15	5
2.1 Software Process Tools and Techniques Evaluation Report	5
2.2 Software Process Management	6
2.2.1 Software Process Management: A Behavioral View	6
2.2.2 Software Process Modeling	8
2.2.3 Software Process Enactment	14
2.2.4 Process Improvement	19
2.2.5 Metrics	19
3.0 STARS IS-15 Task Organization	25
4.0 STARS IS-15 Candidate Tool Acquisition	27
4.1 Constraints on Tool Selection	27
4.2 Tool Selection for Providing a Software Process Modeling Capability	27
4.3 Tool Selection for Providing a Software Process Enactment Capability	28
5.0 The Knowledge-Based Integration Shell	31
5.1 KI Shell View of Process Technology	31
5.2 Summary of KI Shell Features	37
5.3 KI Shell Concepts	37
5.3.1 Method Meta Language	37
5.3.2 KI Shell Process Modeling and Enactment Features	39
5.4 Installation of a KI Shell Application	41
6.0 STARS IS-15 Software Process Case Study Preparation	45
6.1 Preparation and Scoping of "Cleanroom Software Process Case Study"	45
6.1.1 Brief Description	45
6.1.2 Lessons Learned	45
6.2 Preparation of Specification for the CEPA Demonstration	46
6.2.1 Brief Description	46
6.2.2 Lessons Learned	47
6.3 Validation of the "Cleanroom Engineering Process Assistant" Implementation	48
6.3.1 Brief Description	48
6.3.2 Lessons Learned	49
6.4 Major Lessons Learned from Case Study Preparation	49
6.5 Use of Cleanroom Specification Techniques to Model Processes	50
7.0 Software Process Enactment Experiment and Demonstration Preparation	59
7.1 CEPA Demonstration System Description	59
7.1.1 Software Engineering Environments	59
7.1.2 CEPA and Software Engineering Environments	60
7.1.3 CEPA: An Overview	61
7.1.4 Using The CEPA System	62
7.1.5 CEPA Features	65

7.1.6	CEPA Tools	75
7.1.7	Using CEPA (continued)	75
7.1.8	Conclusions and Lessons Learned	86
7.2	Overview of the Process for Developing Process Applications in KI Shell	87
7.2.1	Process for Developing KI Shell Process System Applications	88
7.3	CEPA Prototype System Development Implementation Log Overview	90
7.3.1	March 23 through April 19	90
7.3.2	April 22 through May 3	90
7.3.3	May 6 through May 24	93
7.3.4	May 27 through June 12	93
7.3.5	June 12 through June 21	93
7.3.6	June 24 through June 28	94
7.3.7	July 1 through July 30	94
7.3.8	CEPA Prototype Development Summary	95
7.4	Cleanroom Engineering Process Assistant Installation Instructions	95
7.4.1	Pre-installation Activities	95
7.4.2	Install the Oracle RDBMS	96
7.4.3	Install the KI Shell / CEPA Files	97
7.4.4	Create and Setup the CEPA Account	98
7.4.5	Setup the CEPA FileStore Version	99
7.4.6	Setup the CEPA ORACLE Version	100
7.4.7	Archiving the CEPA Database	101
7.4.8	Restoring an Archived CEPA Database	101
7.5	CEPA Demonstration Operation Instructions and Script	102
7.5.1	CEPA Operation Instructions	102
7.5.2	CEPA Demonstration Script	102
7.5.3	CEPA Demonstration Script	103
7.6	Description of all CEPA Software Source Deliverables	105
7.7	Major Lessons Learned from CEPA Implementation	108
7.7.1	Process Implementation Roles	108
7.7.2	Key Problem and Solution	108
7.7.3	KI Shell's Suitability for Cleanroom	109
8.0	STARS IS-15 Software Representation Work	111
8.1	Software Process Modeling Support	111
8.1.1	Software Process Management System: Overview	111
8.1.2	Software Process Management: Concept of Operation	115
8.2	SPMS: Port Assessment to the IBM RISC System/6000	127
8.2.1	Process Model Database	129
8.2.2	Process Reasoning System	129
8.2.3	User Interface	130
8.2.4	COTS Project Management System	131
8.2.5	Conclusions	132
8.2.6	SPMS: Port Plan for Porting SPMS to the IBM RISC System/6000	133
8.2.7	Schedule	135
8.3	SPMS Prototype System User Training	135
8.3.1	SPMS Training Materials and Discussion	139
8.3.2	SPMS Evaluation Prototype: Hardware Software Requirements	139
8.4	Major Lessons Learned from SPMS Migration Analysis and SPMS Training	142
8.4.1	Lessons Learned from SPMS Migration Analysis	142
9.0	IBM STARS SEE Process Management Architecture Discussion	143
9.1	SPMS Coexistence Strategy with Other Process Management Capabilities	143
9.1.1	HP SoftBench	143
9.1.2	KI Shell	144

9.1.3 SPMS, KI Shell, and HP Softbench Coexistence Strategy	144
9.2 IBM STARS Process Management Architecture Options	145
9.3 Product Integration Strategy	146
9.3.1 The Components for a Process Support Environment	146
9.3.2 Process Support Environment Integration	149
9.3.3 Benefits of a Process Support Environment	151
 10.0 IS-15 Task Conclusions	 153
 11.0 References	 155
 Appendix A. SPMS Training Class Materials	 157

Intentionally left blank.

Figures

1.	The Layered Behavioral Model of Software Development	7
2.	Software Process Management	8
3.	Example of an Approach to Software Development	12
4.	Portion of the RADC Quality Framework	21
5.	Software Quality Framework Factors and Associated Criteria	22
6.	IBM STARS "S" Increment Process Task Team	26
7.	Example of a Concurrent Engineering Process Performed by a Die Designer Role	32
8.	Enterprise Activity Must Be Performed By Using Information / Mechanisms under Appropriate Control	33
9.	The Development and Use of KI Shell Method	34
10.	ACTIVITY & ROLES: To Support a Generic Enterprise Sub-Process	36
11.	PERFORM EVENT CAUSED BY A MOUSE CLICK AT THE USER INTERFACE: Causes Execution of a C Procedure	36
12.	Method Layout for UES's CASE Manager	40
13.	KI Shell's Runtime Architecture	42
14.	Knowledge-Based Shell Will Be Used to Implement a Specific KI Shell Application	43
15.	A Black Box Subfunction from the CEPA Specification	51
16.	Developer Screen Format	53
17.	Using CEPA Facilities to Perform a Black-Box Task	54
18.	Sample Process from Cleanroom Engineering Software Development Process (1 of 2)	55
19.	Sample Process from Cleanroom Engineering Software Development Process (2 of 2)	56
20.	Development Team Leader Screen Format	63
21.	CEPA Administrator Screen Format	68
22.	Software Engineering Manager Screen Format	69
23.	Specification Team Leader Screen Format	70
24.	Certification Team Leader Screen Format	71
25.	Specifier Screen Format	72
26.	Developer Screen Format	73
27.	Certifier Screen Format	74
28.	Using CEPA Facilities to Perform a "Black Box Task"	77
29.	Using CEPA Facilities to Perform a "State Box" Task	78
30.	Using CEPA Facilities to Perform a "Clear Box" Task	79
31.	Using CEPA Facilities to Perform a "Refinement" Task	80
32.	Using CEPA Facilities to Perform a "Correcting Code" Task	81
33.	Using CEPA Facilities to Perform a "Design Certification" Task	82
34.	Using CEPA Facilities to Perform a "Conduct Certification" Task	83
35.	Using CEPA Facilities to Perform a "Submit/Resolve a Question" Task	84
36.	Using CEPA Facilities to Perform a "Failure Report Correction" Task	85
37.	Form for an IDEF0 Process	87
38.	Cleanroom Process Method Layout (Part 1 of 2)	91
39.	Cleanroom Process Method Layout (Part 2 of 2)	92
40.	What Is Software Process Management?	112
41.	High-Level Architecture of SPMS.	113
42.	The System Architecture for SPMS.	114
43.	Project Process Plan Concept	116
44.	A Process Model Component.	117
45.	A High-Level View of Operation of SPMS	119
46.	Process Activity Modeling Symbols	122
47.	Process Activity Links	124
48.	Process Granularity Expansion Concept	125
49.	SPMS Architecture	128

50.	Candidate Trade Options for SPMS Port	134
51.	SPMS Port Plan (1 of 3)	136
52.	SPMS Port Plan (2 of 3)	137
53.	SPMS Port Plan (3 of 3)	138
54.	Hierarchical Structure of SPMS Folder	140
55.	Candidate IBM Process Support Environment Architecture Concept	150
56.	Levels of Integration	152

Tables

1. Primary Alternative Software Process Model Types. 11
2. CEPA Prototype System Development Summary 95
3. Components of a Process Support Environment 148

Products Referred to in This Report

1. AIX is a product of the IBM Corporation.
2. CASE Manager is a product of UES, Incorporated.
3. HP Encapsulator is a product of the Hewlett-Packard Company.
4. HP Remote File Access is a product of the Hewlett-Packard Company.
5. HP SoftBench is a product of the Hewlett-Packard Company.
6. HP/UX is a product of the Hewlett-Packard Company.
7. HP 9000 is a product of the Hewlett-Packard Company.
8. HyperCard is a product of Apple Computer, Incorporated.
9. IBM RISC System/6000 is a product of the IBM Corporation.
10. Informix is a product of Informix Software.
11. Ingres is a product of Ingres Corporation.
12. KI Shell is a product of UES, Incorporated.
13. Macintosh is a product of the Apple Computer, Incorporated.
14. MicroPlanner Xpert is a product of Micro Planning International, UK.
15. NEXPERT Object is a product of Neuron Data Corporation.
16. NEXTRA is a product of Neuron Data Corporation.
17. Oracle is a product of Oracle Corporation.
18. Presentation Manager is a product of the IBM Corporation.
19. PRO*C is a product of ORACLE Corporation.
20. SUN/OS is a product of Sun Microsystems.
21. SUN 3 is a product of Sun Microsystems.
22. STATEMATE is a product of i-Logix, Incorporated.
23. Sybase is a product of Sybase, Incorporated.
24. TEAMWORK is a product of trademark of Cadre Technologies.
25. TEAMWORK/SIM is a product of trademark of Cadre Technologies.
26. UIM/X is a product of Visual Edge Software, Limited, Canada.
27. VAX/VMS is a product of the Digital Equipment Corporation.
28. WordPerfect is a product of WordPerfect Corporation.
29. XVT (Extensible Virtual Toolkit) is a product of XVT Software.
30. XPM is a product of XPM, Incorporated.

Network File System (NFS) is an open industry standard for remote file systems, developed and offered by Sun Microsystems.

Registered Trademarks of Products Referred to in This Report

1. AIX is a registered trademark of the IBM Corporation.
2. CASE Manager is a registered trademark of UES, Incorporated.
3. KI Shell is a registered trademark of UES, Incorporated.
4. HyperCard is a registered trademark of Apple Computer, Incorporated.
5. IBM RISC System/6000 is a registered trademark of the IBM Corporation.
6. Macintosh is a registered trademark of Apple Computer, Incorporated.
7. MicroPlanner Xpert is a registered trademark of Micro Planning International.
8. Microsoft Windows is a registered trademark of the Microsoft Corporation.
9. NEXPERT Object is a registered trademark of Neuron Data Corporation.
10. NEXTRA is a registered trademark of Neuron Data Corporation.
11. OSF/Motif is a trademark of the Open Systems Foundation.
12. Oracle is a registered trademark of Oracle Corporation.
13. Presentation Manager is a registered trademark of the IBM Corporation.
14. PRO*C is a registered trademark of ORACLE Corporation.
15. STATEMATE is a registered trademark of i-Logix, Incorporated.
16. TEAMWORK is a registered trademark of Cadre Technologies.
17. TEAMWORK is a registered trademark of Cadre Technologies.
18. UIM/X is a registered trademark of Visual Edge Software, Limited.
19. UNIX is a registered trademark of AT&T.
20. WordPerfect is a registered trademark of WordPerfect Corporation.
21. X-Window System is a trademark of the Massachusetts Institute of Technology.
22. XPM is a registered trademark of XPM, Incorporated.
23. XVT (Extensible Virtual Toolkit) is registered trademark of XVT Software.

Intentionally left blank.

Intentionally left blank.

Intentionally left blank.

1.0 Document Introduction

The purpose of the "Software Process Tools and Techniques Evaluation Report" is to (1) describe the key activities of the IS-15 Process Task, (2) identify the products produced, and (3) discuss relevant task results.

1.1 Audience of Document

The intended audiences for this document are:

1. The STARS customer: To describe the activities of STARS Task IS-15, to identify the products produced, and describe task results;
2. The STARS Prime Contractors: To transfer technology developed by the IBM STARS team in pursuit of our goal to evaluate, select and experiment with products and techniques to field a software process capability for the IBM STARS SEE;
3. The DoD, Services, and Government Users: To describe software process management technology and tools that can be applied to assist DoD software development organizations in defining, modeling, and testing process models for software development. Further, we will describe for the DoD, Services, and Government users, tools and techniques available today, to implement a process model to facilitate the enactment (or execution) of a project's process for developing software.

1.2 How this Document Can Be Used

1.2.1 Background

There are three major activities towards implementing a software process for enactment in a modern software engineering environment (SEE):

1. The modeling of a software process in a form that can be readily understood and followed; You cannot implement a software process you have not defined and do not understand;
2. The analysis of the process model developed for process implementation and the determination of how processes can be enacted by the SEE or manually;
3. The implementation of the process system, from the process models developed. (By *process system*, we mean a system of processes that have been adapted and tailored to support a selected development or production effort. Further, a process system is built to satisfy stated process driving requirements, such as cost, schedule, and quality goal drivers.)

IBM planned the IS-15 Software Process Management task to provide the IBM STARS SEE, and potentially all of the STARS SEEs, with the ability to support software process modeling, and to illustrate the implementation of a defined software process for enactment. The results of the IBM IS-15 task are presented in this document. Further, this document was prepared to provide the reader with an understanding of:

1. Software process modeling and the *Software Process Management System (SPMS)*;
2. What it takes to implement a well-defined software process model using a commercially available tool, such as UES's *Knowledge Integration Shell (KI Shell)*.

1.2.2 Provide Reader with Basic Concepts of SPMS

This document describes a tool for modeling and testing software processes, called the "*Software Process Management System (SPMS)*" developed under the IBM STARS IR-23/B task. It also describes how the prototype system developed under IR-23/B can be migrated to the IBM STARS SEE and serve as a key component in the planning and modeling of software processes. Further, this document will describe a coexistence strategy of SPMS with other candidate software process management tools.

This document also includes training materials prepared for the SPMS evaluation prototype training class, that were given to the SEI Process Definition Project group to support their software process modeling activities.

From the discussion of SPMS, the reader will gain understanding of how SPMS could be used to support the modeling of software processes. The discussion will also refer the reader to additional source materials for further study.

1.2.3 Provide Reader with a Case Study of Implementing a Defined Process

This document will describe a software process implementation experiment, taking a software process model developed under STARS Task IR-70/E, named the "*Cleanroom Engineering Software Development Process*," and scoping it to produce a "*Cleanroom Software Process Case Study*" problem for implementation in a process enactment tool called KI Shell.

From the discussion of the development and implementation of the "*Cleanroom Software Process Case Study*" problem, the reader will gain understanding of what is required to take a defined process model and implement it using the *KI Shell development environment*, to support the enactment of a selected portion of the "*Cleanroom Engineering Software Development Process*."

2.0 IBM STARS Task IS-15

The purpose of STARS Task IS-15 was to:

1. Evaluate and select technology to support software process definition and enactment support for the IBM STARS software engineering environment;
2. Develop a *concepts of operation* for process definition and process enactment support to determine effective approaches to support software process management for the IBM STARS SEE;
3. Develop a software process and implement it, to examine the software process support tools selected.

2.1 Software Process Tools and Techniques Evaluation Report

The purpose of the "Software Process Tools and Techniques Evaluation Report" is to describe the activities, products and results of STARS Task IS-15. In particular, this report describes:

1. IS-15 activities, products, and results
2. The tools selected to support IS-15 software process definition and enactment experiments and the rationale for their selection
3. The development of the "Cleanroom Software Process Case Study" and the "Cleanroom Engineering Process Assistant (CEPA)" demonstration scenario including:
 - a. The scoping of the "Cleanroom Engineering Software Development Process" for the "Cleanroom Software Process Case Study"
 - b. The development of the "Cleanroom Engineering Process Assistant" demonstration scenario and specification
 - c. The use of Box Structures and Cleanroom specification techniques for representing processes
 - d. The process for implementing the "Cleanroom Engineering Process Assistant"
 - e. The "Cleanroom Engineering Process Assistant" demonstration system that was implemented
 - f. Instructions for demonstration use.
4. The Software Process Management System (SPMS) and
 - a. How SPMS can be migrated to the IBM RISC System/6000 and a plan for its migration
 - b. How SPMS can coexist with other process management capabilities, including KI Shell, the process enactment services provided by HP SoftBench, and cooperative software process enactment support provided by coordination technology.
 - c. How SPMS technology will be employed to support the SEI in reuse-based process modeling and software process asset capture and representation.
5. The SPMS training session prepared for the SEI and the system requirements for using the SPMS prototype.

2.2 Software Process Management

The purpose of this section is to introduce software process management concepts. This section provides the reader with an overview of the state of software process management as well as to identify relevant issues that need to be addressed in future STARS work.

We shall describe software process management from a *behavioral* perspective. We introduce software process management in this way because it is important to recognize that there are many dimensions to the management of process and to communicate to the reader that processes exist -- not only for individuals -- but for groups or teams, and the organizations to which they belong. Next, we shall provide an overview of process enactment concepts. In the enactment discussion, we shall describe the three major views of process that must be modeled to support process enactment, namely the "activity-based process modeling" view, the "role-based process modeling," and the "process information modeling" view. Finally, we shall provide an overview of process improvement and the role of metrics in its support.

2.2.1 Software Process Management: A Behavioral View

A Layered Behavioral Model of Organizational Software Processes

Studies by Walston and Felix <34>, Boehm <3, 4>, McGarry <22>, and Vosburgh, Curtis, Wolverton, Albert, Malec, Hoben, and Liu <33> have demonstrated the substantial impact of *behavioral* (i.e., human and organizational) factors on software productivity. To create software development technology that dramatically improves project outcomes, Weinberg <35>, Scacchi <31>, and DeMarco and Lister <10> argue that we must understand how human and organizational factors affect the execution of software development tasks. Software tools and practices conceived to aid individual activities have been disappointing in not providing benefits that scale up on large projects to overcome the impact of team and organizational factors that affect the design process. The IBM STARS process team has explored concepts to address these problems to bring project and process management closer together.

An extensive study of 17 large software development projects (in 9 companies) done by the MCC in 1986-87 <16> attempted to describe the processes and mechanisms through which productivity and quality factors operate. These descriptions supported our need to understand how different tools, methods, practices, and so forth actually affect the processes controlling software productivity and quality. Since large software systems are still generated by humans rather than machines, their creation must be analyzed and modeled as a collection of *behavioral* processes. In fact, software development should be represented at several behavioral levels <15>. As a result of the empirical studies of software development at MCC, the *layered behavioral model* was created and defined as presented in Figure 1 on page 7 <8>. This model emphasizes factors that affect psychological, social, and organizational processes to show how they subsequently affect process effectiveness, productivity and quality.

The layered behavioral model focuses on the behavior of those creating the artifact, rather than on the evolutionary behavior of the artifact through its developmental stages. At the individual level, software development is viewed as an intellectual task subject to the effects of cognitive and motivational processes. When the development task exceeds the capacity of a single software engineer, a team is convened and social processes interact with cognitive and motivational processes in performing technical work. In larger projects several teams must integrate their work on different parts of the system, and *interteam* group dynamics are added on top of *intrateam* group dynamics. Projects must be aligned with company goals and are affected by corporate politics, culture, and procedures. Thus, a project's behavior must be interpreted in the context of its corporate environment. Interaction with other corporations either as co-contractors or as customers introduces external influences from the broader world of business. The cumulative effects on software development can be represented in the layered behavioral model. The size and structure of the specific project determine how much influence each layer has on the development process.

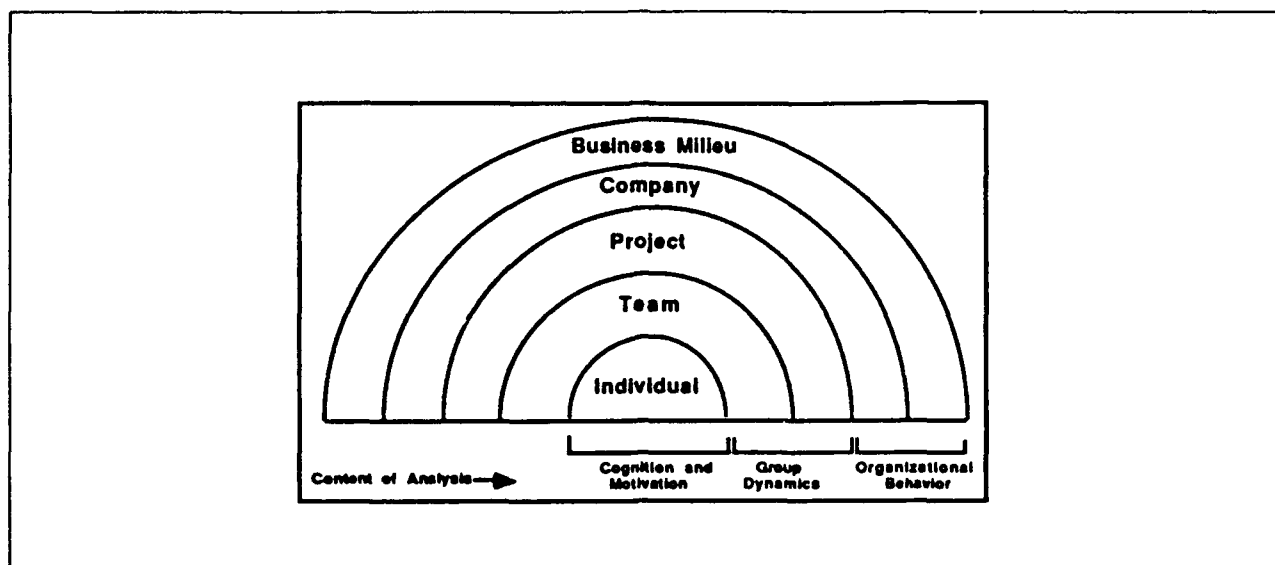


Figure 1. The Layered Behavioral Model of Software Development.

The layered behavioral model is an abstraction for viewing process of large software projects in the context of other behavioral processes. It encourages thinking about a software project as a system with multiple levels of process involved. This does not replace traditional process models of software development, but rather organizes supplementary process views. This is orthogonal to traditional process models by presenting a cross-section of the behavior on a project during any selected development phase. Describing how software development problems affect processes at different behavioral levels indicates how these problems ripple through a project <31>.

The layered behavioral model encourages software process engineers to extend their evaluation of software engineering practices from individuals to teams and projects to determine whether the aggregate individual-level impacts scale up to an impact on programming in the large and allows them to explore multi-project and organizational impacts on projects.

At the project level, qualitative support is primarily process oriented and focuses on who is involved, how to make those involved productive and efficient, how decisions are made, and how tasks are accomplished. This does not undervalue the importance of the product, because the product becomes the focus around which these essentially human design and development processes will be organized.

Therefore, improving software process quality involves understanding the human processes underlying software development <8, 16>, providing methods and technologies to support these processes, and managing them effectively. This is what we mean by *software process management*. (Figure 2 on page 8). Producing a well-designed deliverable becomes the product of a well-managed set of human processes organized around focused objectives and supported properly by process management technology. This involves support for process modeling, simulation, enactment, and process evolution in next generation process-oriented software engineering environments.

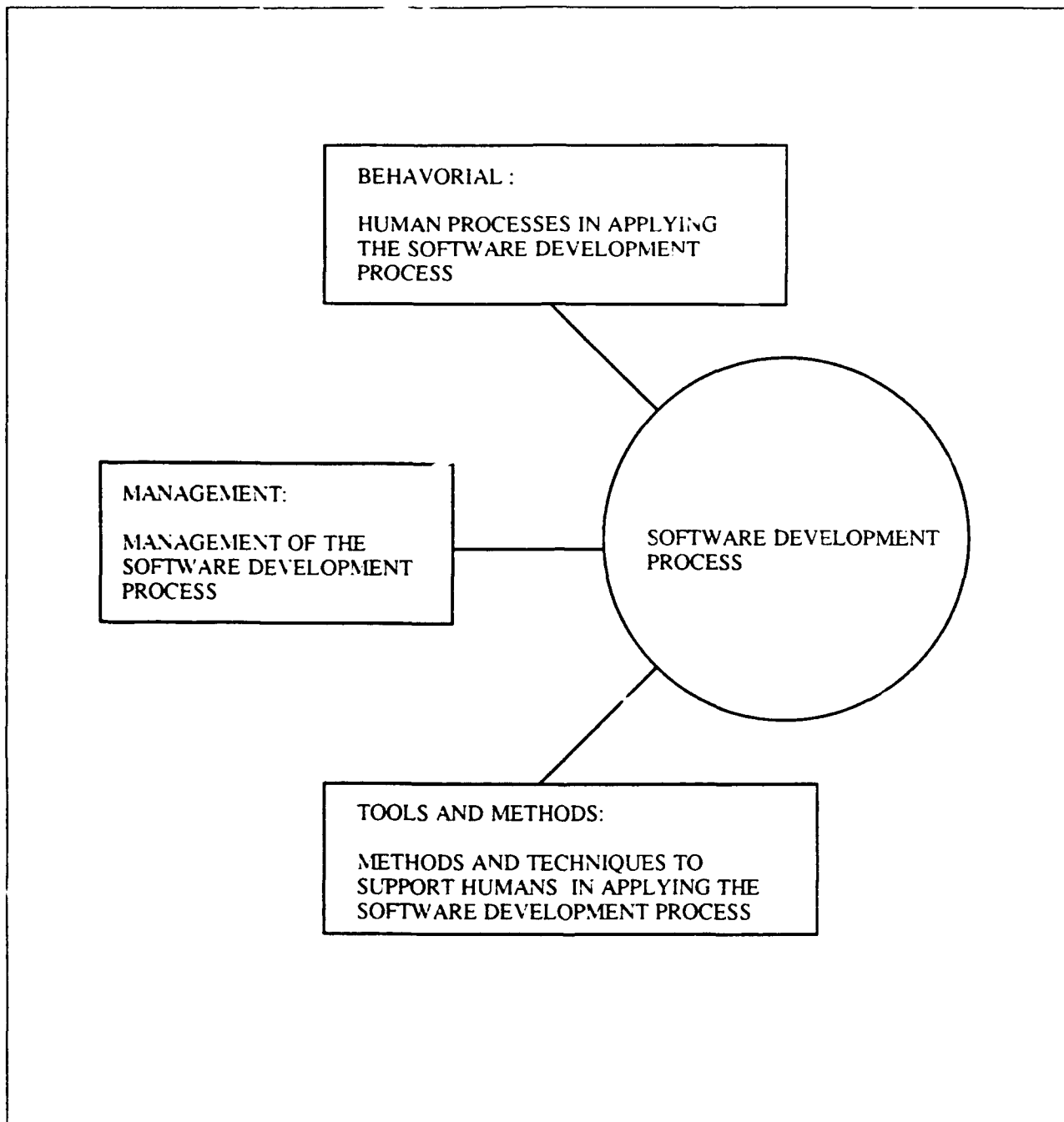


Figure 2. Software Process Management.

2.2.2 Software Process Modeling

Software process research is an emerging area of study within the field of software engineering. To make progress in effectively examining the issues, one must start by establishing a coherent view of the subject matter to provide a conceptual framework for discussion. The definitions presented and the discussion of the interrelationships among them allow the formulation of critical questions about many aspects of the software process.

While the key definitional issues have been discussed at a series of annual workshops (e.g., the International Software Process Workshops 1 through 6), there is no current consensus as to the proper conceptual framework. The view represented here builds on a preliminary hierarchy of concerns, starting at project-oriented

processes that are concerned with establishing the real needs of end users of systems and their impact upon software process models. The view also recognizes the hierarchy of concerns from expertise and technology below all the way down to the structure of particular CASE tools. The organization consists of several levels of progressively more concrete software process issues and concerns.

The primary concern of STARS is how process management will apply to government projects (either contracted or in-house). Therefore, we focus our discussion at the project level (i.e., project-oriented processes), which serves to bound the scope of considerations as seen in the layered behavioral model. The broader process concerns of a multi-project line of business are discussed later in section 2.

2.2.2.1 Project-Oriented Software Development Process Modeling

The primary focus of this section is to describe the domain of software development process models. The term *software development* is taken to encompass the full range of software and system-related activities from problem statement through post delivery maintenance, modification, and evolution¹ (i.e., cradle to grave, life-time, or sometimes life cycle). There are other views of software engineering practice that do not focus on process issues (e.g., an information-oriented model), and these must be considered as well. In the following paragraphs, we define the terms of our conceptual framework and illustrate the definitions by posing relevant issues at that level of concern.

Considerable confusion has arisen in past discussions of the software process because of a lack of common definition. Therefore, we define the following terms.

Software process:

The collection of related activities, events, mechanisms, tasks, and/or procedures seen as a coherent process involved in the production and evolution of a software system that satisfies a given need.

Software process model:

A descriptive representation of a software process that supports explanation, reasoning, simulation, and so forth. A software process model should represent attributes or views of a range of particular software processes and should be sufficiently specific to allow reasoning about them.

Software process models are valuable for supporting effective process management involving aspects of: process planning, enacting, predicting, monitoring, adapting, and correcting. Software process models allow for the:

1. Enabling of effective coordination by facilitating the communication of a formalized process leading to a consensus understanding across the organization or project.
2. Enabling of process reuse by facilitating analytic selection of a process model from a base set of alternatives that include components and process abstractions. Reasoning about the alternatives is also enabled.
3. Support for process evolution, adaptation, and correction occurs owing to the ability to define model-based process measurements, experience collection, and process rationale.

Relevant issues in process modeling at this level of concern are:

1. To what extent is a model descriptive or prescriptive; that is, how does it correspond to (correctly describe) how software is really built or to an ideal of how software *should* be built?
2. To what extent does a model describe the entire process as opposed to some aspect or view of a process?

¹ Evolution implies that system functionality evolves over time.

3. To what extent is a model useful in developing managerial and technical approaches to software development? Managing these extents (e.g. the gap between the real and the prescribed ideal process model) is one of the duties of the new process engineer.

2.2.2.2 Types of Project Process Models

Simplistically, a large software project occurs within the context of an ongoing negotiation about what the user wants, what the customer can afford, and what the developer can build. This occurs within a management context that must deal with three principal factors: cost, schedule, and quality. Within that context there exists a range of possible strategies for defining an organized software development process on a large software project. This range is delineated by two extremes. On one end of the spectrum is the *job shop* philosophy, in which each project requires a totally different process based upon specialized project criteria. This seems to be the currently predominant strategy within DoD projects. On the other end of the spectrum is the *factory* philosophy in which a standardized process is designed in advance and is reused on each project. This has been used successfully by the Japanese software factories, and the advantages gained have been described by Cusumano <9>. In the middle of the spectrum would be the *semi-factory* philosophy in which parts of the process are reused on each project within an adaptable framework addressing specific project needs. We expect DoD projects to continue to be in situations where process adaptability is important. Therefore, we present the following set of high level process model types that can be reused on projects.

The four primary alternative project-level software process model types are:

1. Waterfall,
2. COTS adaptation,
3. High-level specification transformation, and
4. Exploratory/incremental (includes Spiral).

(See Table 1 on page 11.) We classify each model according to the situations in which it is most applicable and provide example applications for clarification. We also discuss how to determine, or select, an appropriate model in the context of a contract program. It should be noted that these models can also be used at the subproject level as well, and it is appropriate for more than one of these types to be used on a large project, leading to a higher level model that contains these as subprocess instances. The integration of subprojects using different process models is currently problematic. A fifth type is the widely used, ad hoc (i.e., undefined) process model -- a discussion of which is beyond the scope of this section. It is not clear, as yet, whether these model types are instances of any kind of process metamodel from which these inherit properties (an interesting research topic).

The waterfall model has been described in many publications (e.g., Royce <30>); its applicability, however, has not. We see its primary usage in those situations in which the functionality, architecture, and technology are well understood. Examples of this might be inventory control or data reduction and reporting. The COTS adaptation model is most applicable in situations in which the application and technology are well understood and where there are several commercial packages available that together achieve almost all of the functionality required. The added value becomes the adaptation and integration of those COTS systems. Example applications are financial management or document preparation. The high-level specification transformation model is most useful when the functionality required is not well understood but where 4GL technology is available for the creation of high-level "programs." Example applications include MIS or forms management. The exploratory/incremental model is most useful when the major application issues are not well understood (e.g., feasibility, usage scenarios, functional features desired, etc.) and when a movement to more stable capabilities is desired. This might include one or more early prototypes to clarify these issues. Example applications include military C3I, SDI, or real-time embedded systems.

	Process Model Name	Situation Characteristics	Example Application
1	Waterfall	Well-understood application, architecture, technology	Data reduction
2	COTS adaptation	Same as waterfall, but several commercial packages available	Inventory control
3	High-level specification transformation	Ill-understood application, use of 4GLs	MIS
4	Exploratory / incremental / spiral / rapid-prototyping	Major application and technology issues not understood; need for early subset demonstrations	Military C3I
5	Ad hoc / undefined	???	???

Table 1. Primary Alternative Software Process Model Types.

The major issue relative to choosing one of the model types is in how to interpret the notion of "well understood" - that is, by whom is it well understood? We believe that this criterion must apply to all stakeholders in the process if the model is to be used and adapted effectively across the system's life. Major techniques to be used in the exploratory/incremental model are prototyping and risk management. We define these techniques as those that increase the probability of producing a useful, fieldable, and supportable system.

The relationship of these model types to the military acquisition, procurement, and contracting process can be identified. During technology base, demonstration/validation (DEM/VAL), and proof of principle efforts, for example, the exploratory/incremental model is recommended. During the transition to full scale engineering development (FSDE), an evaluation must be made to see whether one of the others should be used, with the default being the exploratory / incremental model rather than the waterfall model. During FSDE, if the technical approach tends toward heavy use of COTS, then negotiable requirements are achieved by using the SOW requirements as COTS evaluation criteria, with selected COTS redefining the requirements. If a point is reached in which all requirements, technology, and architectural issues are well understood, then a waterfall approach is justified. The choice of H/L specification transformation depends on the availability of 4GL (or automatic programming) technology within well-defined application domains. This approach is only recommended for traditional business-type applications or until domain modeling techniques and automatic programming techniques come together for very specific application areas in the future. Strategies for exploratory/incremental model contracts include the use of evolutionary software development processes (SDPs) SDPs and issue/risk management plans. During the transition from FSDE into installation/fielding/operation/support, we recommend that the process model be transferred as the maintenance process starting point. This needs to include process history and rationale to be effective on a continued system evolution basis.

There are a number of applicability questions to consider.

1. Are there empirical reasons for preferring one model to another; that is, do they lead to more cost-effective software development and/or a higher quality product?
2. Can there be any a priori reason for preferring one model to another? If so, what criteria are relevant?
3. Can a given technical approach conform to more than one model?

Current wisdom asserts that a given technical approach (and the process model it conforms to) might be more appropriate for: a particular class of applications (application domain specificity), a particular class of system architectures (structural specificity), or a class of organizational structures (organization specificity).

The relationship of project process models to design approaches, methods, techniques, technology, and tools is such that the process models are supported by these more specific items in specific situations.

2.2.2.3 Approaches, Methods, Techniques, and Tools

In support of project-oriented process modeling, chosen approaches (and subsequently methods, techniques, and tools) carry out the objectives of that process model and are mapped onto one another. For example, the developmental project approach in Cleanroom describes the separation of requirements, development, and certification concerns inherited from the goals and objectives of the Cleanroom Engineering Life Cycle Process Model.

Approach to software development:

A strategy for achieving the development of a software system in a way that conforms to some software process model. An approach can be expanded into a more detailed approach.

A simple (highly abstract) example of an approach to software development is shown in Figure 3.

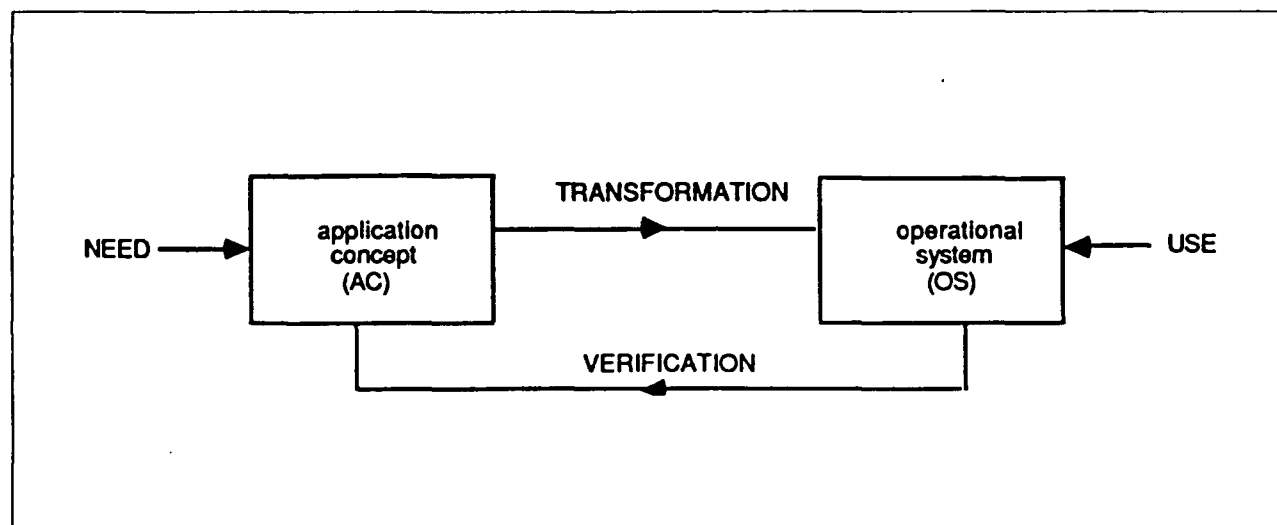


Figure 3. Example of an Approach to Software Development.

The example in Figure 3 immediately raises a number of questions about the representation of the model (both syntax and semantics). For the graphical syntax questions arise about the meaning of a box, a line, and an arrow. What are the rules for legally combining these symbols? What are the meanings of the terms *application concept (AC)*, *operational system (OS)*, *transformation*, and *verification* used in the approach? For example, AC might be a set of user scenarios or a requirements document; OS might be a functional prototype. The term *transformation* might mean development and *verification* might mean testing the OS against the requirements. The terms of the approach (which is also a model in the general sense) can only be defined in the context of the broader process model goals and objectives.

Software development methods and tools support the application of approaches to software development.

Method:

An explicit prescription for achieving an activity or set of activities required by an approach to software development.

Methodology:

A study of methods. The term is often used incorrectly as a synonym for method.

Technique:

A systematic procedure by which a software engineering task is accomplished. Typically, a technique is considered to be supportive of and subordinate to a method.

When looking at a specific method, further questions are in order: With what approaches is the method consistent? How good is the method when it is compared with other methods consistent with the approach? How much of the process does the method prescribe? How can the method be automated - that is, supported by a software tool?

Software tool:

A program or collection of programs that can support the application of a method or technique.

Although this definition does not exclude general-purpose tools, it does insist that tools can only be judged by the extent to which they support methods, for example, CADRE Teamwork's support for the DFD (dataflow design) method. In addition, we are concerned with usability and efficiency. Is a given tool easy for the intended user to use? Is it efficient compared with some other support for the same method? While individual software tools are of undoubted value, coordinated collections of tools are attracting increasing attention. These are termed *software development environments*, a term taken to be synonymous with the terms *programming support environments*, *project support environments*, *software engineering environments*, and *integrated programming/project support environments*.

We take the position that an unstructured bag of tools does not qualify as a software development environment. Thus, we define the term *software development environment* in the following way.

Software development environment:

A coordinated collection of software tools organized and adapted to support some approach to software development or conform to some software process model (sometimes called a software engineering environment or SEE).

A software development environment (SDE) is useful to the extent that it is an improvement on an uncoordinated collection of tools. It is possible for an SDE to support more than one approach or conform to more than one process model.

Multi-Project Process Management: Further project-specific characteristics of interest are: visibility of key subprocesses (observable), a basis in frequent and evolving demonstrations of emerging functionality, continuous customer/user involvement, iterative/incremental development, early participation of key life-cycle stakeholders at all times, tailored appropriateness to specific development situations, specification following implementation, and flexibility over the life cycle. Several related technologies are emerging to support a quality-driven process (e.g., concurrent engineering, process management, intelligent groupware, AI-based design, prototyping, and incremental development). When brought together with the goals of a quality-driven process, these technologies can lead us to define a class of new software life cycle process models used in the projects.

When a project process is viewed in the context of a total business process, a software development project becomes an instance in the growth of a base of software assets. The basic principles of such a multi-project, *process-driven, software business philosophy* are continuous software process improvement, process knowledge definition, user/customer focus, commitment to software quality at all levels of an organization, teamwork, investment in people, reuse, and, most important, customer/user satisfaction.

2.2.3 Software Process Enactment

The purpose of this section is to discuss languages and notations in which formal models of software processes could be represented and then enacted. The word "enacting" has been used instead of alternatives such as executing or interpreting. We wish to preserve the concept that the mechanism for running process models is a symbiosis of human being and computers, and at the same time not to hint at particular roles for either partner. Executing has strong connotations of machine execution; interpreting can denote activities in man and machine which are very different; enacting was chosen as a neutral and previously unadopted term.

Enactability simply means that human beings involved in the software process receive computer guidance and assistance in what is an extremely complex activity. Put another way, models are not just used "off-line," as a means of studying and defining processes, but also "on-line" while processes are being carried out, as a means of directing, controlling, monitoring, and instrumenting them. To help clarify what enactment is not, we assert that it is neither (a) writing programs that wholly mechanize software production, or (b) writing programs that wholly prescribe what human participants in the process are to do.

Process enactments are written to define possible (allowable) patterns of behavior between non deterministic human beings and systems constructed of computer programs. Modeling and programming the software process is an experimental testbed for modeling and programming human-computer activity in general, for introducing a new and potentially much more highly productive way of software system-building.

2.2.3.1 Enactment Formalisms

Language is a central issue. There is a "horse and cart" problem here: to find out what language features we need, we need to write enactable process models; to write creative models, we need the appropriate language features. An issue is that tradeoff between the expressive power of a notation and the ability to reason about or analyze the process model beforehand. A language design tradeoff involves subsetting expressive power of a notation to prove some useful properties about the process, or, conversely, providing more expressive power but limiting the ability to reason about the process.

A prescriptive interpretation of a Process Enactment Language is essential since it is desirable to use the language to express plans (i.e., descriptions of processes that are intended to take place in the future). Prescriptive means that the process is laid down as a guide or rule of action. Plans can either be followed by choice or enforced, but they are prescriptive and sometimes proscriptive (that is those things that are forbidden are described as well). The prescriptive interpretation of a language requires some form of interpreter.

2.2.3.2 Enactment Architectures

This section focuses on the mechanisms needed to enact software process models from the viewpoint of three key aspects of architectures: information, operations, and operators (i.e., entities that apply operations to information). The interactions of these aspects, of course, are important as well. The operations view of process modeling is a traditional activity of Project Management developing a network of activities for their project, at an appropriate level of granularity. The operator view of process modeling takes the personnel roles that will be required for a development effort and allocates the activities or process tasks identified to appropriate personnel roles. The process information base view associates the data objects that both the activity process view and the role process view require to permit process enactment.

There is no single *best* approach for modeling processes for enactment, but to enact a process, all three views of process are required. A logical approach to process modeling is to perform "activity-oriented" process modeling to identify threads of project activities along with "role-oriented" process modeling to allocate these activities to project roles, while at all times identifying required data objects. It is probably best to develop process models incrementally and to plan to examine all three views within each increment.

Process Information Base: When enacted, executing process models must manipulate information. The information differs from that needed for product programs in content, in structure, and in the properties of the supporting operations. The information base required to support enactable process models includes three separate kinds of information. The first comprises the process data, which consist of information relating to the execution of the process model (e.g., lists of project activities to schedule or plan). The second is process state, which consists of the internal state of the executing process model (e.g., a process program counter). The third comprises the product data, which consist of the project documents (e.g., schedules, budgets, module specifications, code, etc.)

The list of information base requirements includes: objects of varying sizes, varying degrees of persistence, nested transactions, very long transactions, complex and programmable relations among objects, triggering mechanisms, automatic inferencing mechanisms, dynamic types of schemas, multi-user sharing with associated locking mechanisms, versioning, powerful query languages, partitioning and view mechanisms, and tolerance of inconsistency. An information base for process models must also support multiple, dynamic, programmable notions of consistency.

Operation Classes: The execution of process models requires several classes of operations. The two most familiar operations are control flow and composition. Control flow is a well-understood notion. Composition means arranging lower-level elements into a higher-level structure, for example, as in the way P1074² process fragments are composed into a process model. Two other operations -- instantiation and instrumentation -- are both essential to enacting process models and different from operations available in most programming languages. Instantiation is binding an abstract notion to a concrete instance. In programming languages, abstract data types are instantiated to concrete representations and implementations, programs are instantiated into executing processes, etc. Instantiation is more important in the context of enacting process models. Instrumentation is the way in which mechanisms are applied to carry out the execution. Instantiation can also be taken to mean the way metrics are attached to measure process attributes.

Instantiation is necessary to get the dynamic and flexible characteristic demanded by executing process models. Many aspects of process models require instantiation: activities can be instantiated in various ways, depending on whether they will be performed manually by people, automatically by tools, or semi-automatically by a combination of people and tools; the next event to execute can be dynamically instantiated according to the current state of the process and (perhaps) directives by users; unusual conditions can be handled by instantiating exception handlers; and so on. Second, instantiation need not be complete, as in most product programming languages. This is so in part because of the dynamic nature of executing process models and in part because of the expected long execution time of process models. The dynamic property demands instantiation because not all decisions can or should be made statically.

Operator Types: Operators are the entities that apply operations to information. For product programs only the hardware (or a virtualization of hardware) actually performs manipulations. For process enactments however, two operators exist: (1) the real and virtual hardware and (2) people. It is perhaps this added operator that most clearly distinguishes enactable process models from product programs. People must be considered to be operators because there are many fundamental operations that must be performed that cannot be performed automatically. For instance, during execution of a process enactment, a manager may be "invoked" to select a particular schedule or to bind a particular programmer and task together. Some of these, such as defining and selecting a schedule, can take a long time and can have significant consequences; hence, the notion is really quite different from simply entering data to a product program.

² The IEEE-P1074 standard for software lifecycle process is a draft standard which was prepared by the Software Life Cycle Processes Working Group of the Software Engineering Standards Subcommittee under the sponsorship of the Technical Committee on Software Engineering of the IEEE Computer Society.

2.2.3.3 Enactment System Requirements

Ideally, to support the manipulation of related process model aspects, a system for process enactment must contain support for the following capabilities:

Products: identifiable bodies of information, describable by decomposition to primitive types, or in terms of a class system (inheritance, specialization).

Activities: the transformation of inputs to outputs, describable in terms of inputs, outputs, preconditions and postconditions; conditions may refer to some global "state." Conventional "project management" features, such as durations of activities and date information, tends to be variable, and means to express variability need to be part of the language.

Agents: things that perform activities; these may be human or computer-based; particular human agents may operate in a number of different roles; agents may have different "views" of the process. Presumably the preconditions of an activity may refer to the capabilities of an agent, but there has been little thought of how this might be represented. An activity description needs to indicate the capabilities required of the performing agent; such requirements need to be related in some way to the functional definition of an activity. Allocating agents to activities requires a "type system" for resources and should consider what capabilities would be needed for agents (determined by looking at "natural" processes and the human use of tools).

Control flow: Composition of activity instances using conventional notions of sequence; selection and iteration or Prolog-style backtracking.

Communication: Synchronization of activities and transfer of products between agents; notions of dialogue and commitment between agents. Modeling dialogue and cooperation between agents appears to provide an alternative approach to techniques such as dataflow and state-transition modeling. Coordination technology may serve as a useful approach here.

Decisions: Choices made in the light of some "goal"; creative actions. The notion of "goal" support needs more investigation. We may well have goals that relate to properties of the process itself (such as duration, resource utilization, delivery schedule, public relations), as well as goals that relate to properties of products. The way that decisions are made in the light of goals and the non determinism that this implies need to be looked at.

Long-term execution: Process enactments are expected to run for long times and they are likely to change while they are running. The effects of persistence and change are not currently well understood, and there is little in current languages or programming systems to provide guidance. For example, if the enactment changes, how is a trace of execution from the old version to be interpreted.

Concurrency and communication: Process enactments will be highly concurrent, and communication among agents and processes is a central issue. This will require more support than is currently available.

Nondeterminism: Although process enactments will have to deal with the traditional form of nondeterminism (i.e., having several things that can be done at a time), the greater source of nondeterminism will be due to the human factor.

Views: Multiple views or representations of process models will be necessary to avoid the problem of having to read the "code" to understand the process. Three different types of views are needed: (1) views as projections that present certain information to the user while hiding the other information, (2) views as different ways of representing the same information to the user (e.g., graphical and textual), and (3) views to support navigation (e.g. scoping) over a defined hierarchy of levels of process abstractions.

The term "view" has been used in two different ways: (1) to mean the restriction of an agent's "domain of discourse" to some part of the process (this is like the database notion of view), and (2) to mean the different "dimensions" of a process - for example, technical, organizational, and managerial dimensions.

Using different views as an aid to understanding the process implies that it should also be possible to use those views to change the model when necessary. The problems with manipulating base table data via

views are well known in the database community. Working backward from a modified view to change the base information can be very difficult. An alternative is to store each view explicitly, but there are still problems in maintaining consistency among the various views.

Role Support: It has been proposed that there is a close relationship between role and view, with a view plus a role defining a "virtual agent." Important concepts are: role "type" (i.e., manager), role "instance" (i.e., the manager of a particular project), and role "occupancy" (i.e., the binding of an agent to a role instance). Further, the role is a way to assemble a view of the processes (tasks or activities) for which a particular role type is responsible.

Rules: Rules can be used, in one form or another, to specify what is to occur. Triggering (i.e., when do rules actually fire) is an important issue to be explored. Accessibility of the rules for understanding the process is also an important issue. Both static and dynamic accessibility are necessary. Static accessibility involves understanding individual rules. Dynamic accessibility involves understanding what actions a given set of rules produce or, conversely, what rules were used to produce a given action.

Sharing and containment: Complex documents may contain many objects, and an object may be shared among several different documents. This poses significant problems for multiple agents who share access to these objects and have the ability to make changes in them. Traditional database notions of consistency and commitment are not likely to be sufficient.

Hierarchy and decomposition: The (possibly concurrent) existence of subactivities and their parents will require special language features. For example, it may be necessary for a process or activity to "know" that it has a parent or concurrent siblings.

Types: Process enactments will require rich type systems to describe the various artifacts produced in the software process as well as the relationships between them. It is not the artifacts themselves but the relationships between them that provide meaning, and it may be desirable to include relationships among the types in the type definition. Language features for process enactment will also need to support type evolution -- changes in the definition of the type during the process.

Extensibility: A programming system for process enactment will need the ability to accommodate new ideas and new notions of process as they change.

Reuse: What does it mean to reuse a process enactment and how do we do it? A number of potential approaches to reusing process components include: development of proper abstractions, appropriate levels of granularity, late binding, and development of "generic" programs or program fragments. Reuse should not be limited to the code process enactments; specifications and designs are also potentially reusable.

Process change: The issue of process dynamics is concerned with aspects of how things change. There are several degrees of change in the process model life cycle: changing instances, changing types, type systems, etc. Another important aspect is that of process generation and process modification while the process is executing in real time, either by self-modification or external modification.

Dynamism is the ability to change the enactment while it is executing. In a model without dynamic character, a trace of execution of the model would contain only components that were built into the original model. In a dynamic model, an execution trace might include components that were not in the original model. The minimum amount of dynamism required to support process enactment is type instantiation and process instantiation. One way of handling changes is to delay the binding or instantiation until as late as possible.

Several questions related to dynamism in the software process are unresolved. What is the source of dynamism in process models? Modifications might, for example, be made in response to exceptions, or they might be related to higher-level goals such as optimization of the process. Should process enactments be self-modifying, modified by humans, or both? Self-modification can be dangerous; there is the need to handle certain conditions (e.g., exceptions) automatically. Two related questions are: How much dynamism is desirable in a process enactment? How much is necessary?

2.2.3.4 Engineering Enactable Models

The following tradeoffs must be considered in developing enactable process models:

1. A suitable compromise must be found to settle the tension between what should be included in the enactment mechanism and what should be in the associated process information base.
2. The "proper" level for the supporting enactment mechanism must be determined.
3. Instantiation and instrumentation are requirements for any enactment mechanism.
4. The requirements of the information base must be carefully defined, and must, in particular, permit specific forms of inconsistent data to be retained.
5. The problems of how people interact with enacted process models are key; this includes the social issues, the form of the human-computer interface, buy-in, cultural fit, etc.

There is a life cycle for software processes analogous to the life cycle for software products. Requirements are formally or informally defined, the processes are designed through several levels of refinement, process enactments are constructed, analyzed, tested and debugged, processes have components, versions and configurations that evolve over time, and process components may be reused across a range of software and other kinds of development projects. There are meta processes that guide this life cycle.

Requirements: Deciding what can be expressed in a process model and what cannot happens during the requirements phase. This may be with respect to a particular formalism selected *a priori*, or it may be possible to put off formalism selection until the design or even construction phase. Certain requirements cut across essentially all processes, such as hierarchy, concurrency, and nondeterminism. Some requirements simply cannot be met within a formalism. One view holds that process models are primarily theoretical tools for understanding processes, so computability, decidability, and so on are not prerequisites. Another seemingly opposing view holds that our understanding of some phases of the software process is too immature for modeling - in the context of actually enacting the models.

Design: Devising a means, within the chosen formalism, for meeting the requirements happens during design. This includes breaking the overall process into subparts (sometimes referred to as architecture), developing data and process models, etc. Trace the history of actual projects and then design models by generalizing from the components of these traces. The State Change Architecture approach < 26 > provides components of a model statically, but the links between the components are constructed dynamically during enactment of the process. Process prototyping merges design and construction phases to some extent by advocating, with a distinct concern for evolution and controlling the impact of change.

Construction: Actually writing the process model or program within some formalism is done during construction. We can imagine the application of various programming tools such as language-based editors, cross-reference tools, etc. (e.g., Aspects' use of windows, views, and defaults.) Artificial intelligence technology can be applied to reasoning about a process with respect to its "purpose," that is, the goal of the process in the particular context. Other advanced tools, including specification generators and analyzers, functional simulators and hypertext can be employed. For large processes there will be models that have to be composed. Enactable models are then translated into some internal form appropriate for the formalism. The notion of a software process architecture has been defined to support adoption of common technology across processes.

Testing and Debugging: Testing can be done via simulation; e.g., an activity coordination assistant that provides a testbed for new policies. Alternatively, the only testing may be by use in actual projects. Debugging can be done off-line while a project is in progress. A tool for experimenting with process programs could be developed, in the form of dialogues, by running them backwards and forwards, directly modifying their stores and so on.

Evolution: Processes will evolve over time. The problem of change propagation exists: updating the behavior of software development environments and the contents of software artifact databases to reflect changes in enacted process models. This is a distinct problem that deals with actual conversion of existing artifacts and perhaps existing tools, as well as the retraining of users.

Reuse: Process modeling would not be very effective if new processes had to be constructed for every software project. If one views the environment and the model as orthogonal, this may lead to greater opportunities for reuse since the model is not tied to a particular environment. Reuse activities include selection, adaptation, assembly, cataloging, and assessment that apply equally well to reuse of software components and reuse of subparts of software processes.

2.2.4 Process Improvement

By defining processes for developing systems, of which software may be a critical part, an organization possesses a tool to ensure great consistency in the way systems and software will be developed. When an organization has defined, at an organization level, a process for supporting software development, the organization has reached the level of 3 on the SEI Software Process Maturity Capability Model. However, as the definition of processes for developing software is a human activity, defined processes may be suboptimal and in need of improvement. Further, a good process may deteriorate over time if it is not continually examined with respect to the technology base to support software development or the changes made in the corporate culture that affects software development. It is possible to define a suboptimal process that may have a less-than-positive impact on the organization that uses it, possibly worse than having no process defined at all. Thus, there is a need to constantly monitor and analyze processes to ensure that if they are in need of improvement, they will be improved. Processes can only really be tested through use. Some tools such as the Software Process Management System (SPMS) permit process engineers to analyze the performance of a defined process, before its deployment for use. However, user experience and measuring the results of the use of process enable us to improve the process.

2.2.5 Metrics

Quantitative measurement of the results of processes, and the processes themselves, provides the basic ingredients for studying trends and explaining qualitative observations about processes such as "I don't know exactly what is wrong with this process, but it appears to waste a lot of my time, and just doesn't work very well for me." Lord Kelvin is attributed with expressing the idea that 'if it cannot be measured, it cannot be improved.' By establishing metrics, associating them with an organization's process, and analyzing them for indicators where processes require improvement, an organization can achieve an SEI level 4 on the Software Process Maturity Capability Model.

There are two basic categories of metrics that assist process engineers in analyzing and improving processes, namely, product metrics and process metrics. Product metrics are a measure of the quality of a product produced by a process. Thus product metrics are an indirect indicator of how a process is working. If the product being produced is suboptimal, so must the process be that was used to produce the product. Process metrics are measurements established and taken directly from the performance of process tasks. In this way, process metrics are a direct indicator of process performance. We shall briefly describe a scheme for product metrics and a scheme for process metrics.

To prevent metrics analysis from becoming a draconian management tool, metrics, both product and process, must be collected and analyzed at an organization level. Data collection must be geared to collecting aggregate trends and analyzed as such, and management misuse of them must be avoided.

2.2.5.1 Product Metrics

Product metrics are used to examine factors associated with the products that can be analyzed to identify aggregate trends to isolate and correct process problems. Key to effective product metric collection and analysis for process improvement is the development of an effective measurement model that identifies the criteria for metric selection and the way the metrics will be employed to support process improvement. Some of the requirements for an effective measurement model < 38 > for a process management capability include the following criteria:

1. Measurement coverage should include the architectural levels required by the executable process model (e.g., System, CSCI, CSC, CSU).
2. The model should provide for aggregation of measurements from the low levels of software components to larger components.
3. Measurements should cover phases of the life cycle (e.g., design, coding, etc.).
4. Measurements should cover documentation and software components.
5. Measurements should be related to intuitively meaningful concepts of quality.
6. Measurements must be explicit so that they may be represented in process models.
7. Implementation of the measurement model must be tailorable and extensible.
8. Ideally, historical data concerning the values of the measurements should be available to aid in their interpretation.

The term *measurement model* refers to the relationships of individual metrics to the software development process.

Measurements must be available for each level of the software component (e.g., CSCI) that is to be created by the executable process model. This enables the use of these measurements in the validation task, which evaluates the success of the process. Because the quality of a software component is partially determined by the quality of its parts, one must aggregate the values of measurements of the parts to assist in determining the quality of the larger component.

Measurements must also be available over the domain of the software component. Different measurements are appropriate at different points in the life cycle of a software component. Documentation is a major deliverable item in software development. The quality of this deliverable has a major impact in the long-term costs and quality of the software product. Measurements of the quality of the documentation must also be available within the measurement model. The measurements in the model must be related to intuitive concepts of quality.

Reusability plays a large role in the STARS SFILC process model. Other intuitive concepts, such as portability, modularity, and generality, are related to reusability.

The measurement model must be related to abstract notions so that one may intuitively grasp what is being measured.

Even though the measurements must be related to abstract concepts, the measurements themselves must be explicit so that they may be represented within the process model. This implies that very specific questions or measurements must be present in the model rather than very general questions (e.g., Is the CSC free of microcode instructions? vs. Is the CSC of high quality?). The measurement scale may vary from nominal (e.g., yes vs. no answers) to numbers along a continuous scale. It is important that the questions and the formulas be known and available for use by the process model.

As Ross has noted <29>, metric collection can drive model refinement. It is likely that as the process models evolve, the measurement model will also evolve. The implementation of the measurement model should be extensible and tailorable so that this evolution may take place.

The RADC Quality Framework: The *RADC Quality Framework* <40> describes a quality model in which a hierarchical relationship exists between a user-oriented quality factor at the top level and software oriented attributes at the second and third levels. Figure 4 represents a portion of this model with the factor efficiency. Figure 5 on page 22 shows the relationship of the factors and criteria in the framework. The metric elements (the specific questions applied to a project under development to assess and predict quality) are applied at various levels of software architecture and at various points in the software development life cycle. By using the framework, one can define the data that must be gathered on specific architectural units at specific points in the life cycle. The framework provides a series of formulas that relate the data to specific criteria to be measured and finally to the actual factors themselves. The factors that can be selected by a user and their relationships to various criteria are shown in Table 2.

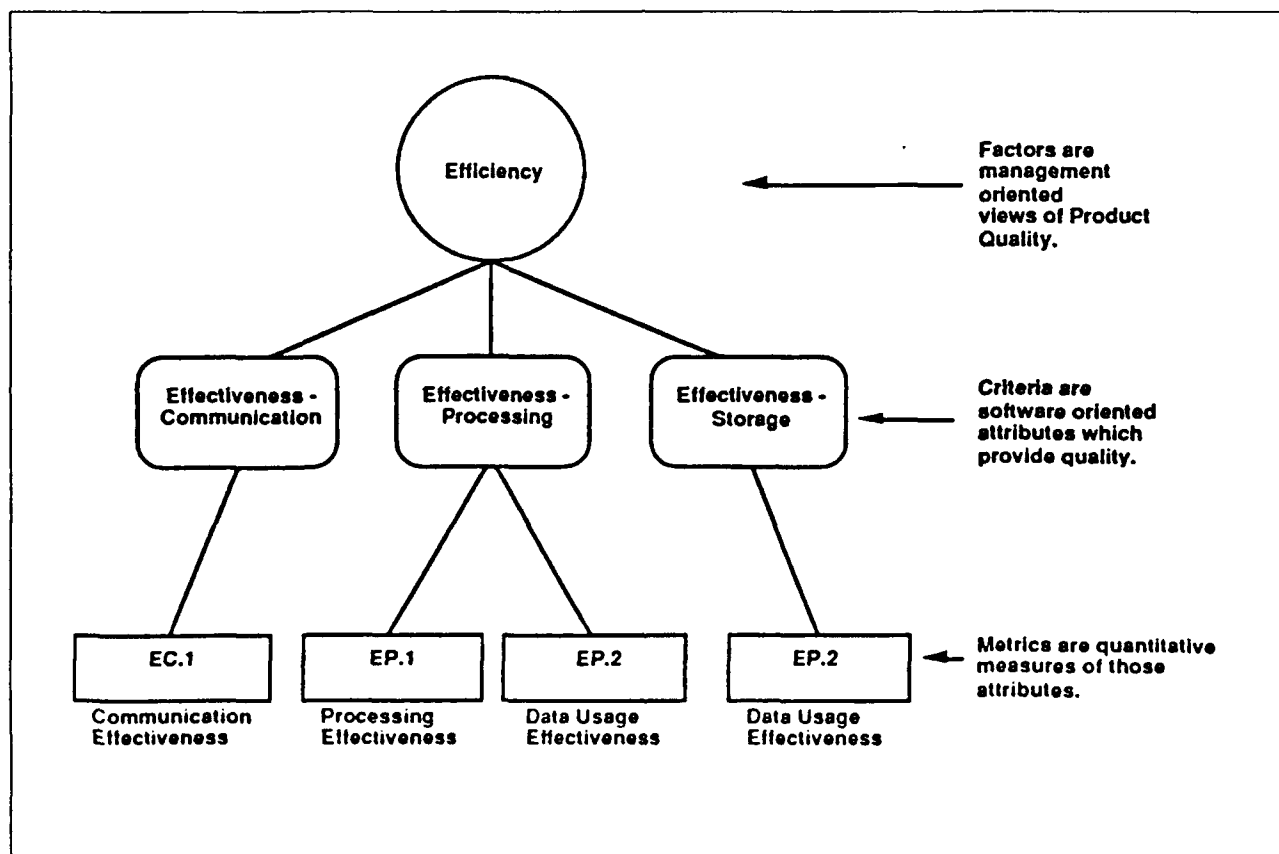


Figure 4. Portion of the RADC Quality Framework.

The RADC quality framework, selected for use in SPMS, is based on the need for a measurement model and the belief that the factors and criteria represent reasonable concerns in the software development process. There is no clear evidence that the specific measurements themselves are optimal, nor is there any evidence to indicate what particular values of specific measurements should give one confidence or indicate an area that should be of concern. Although the RADC Quality Framework does not meet the requirements for a measurement model discussed in this section, it is extremely useful in providing an initial set of measurements that could be used in relation to executable process models. The RADC Quality Framework is explicit and hierarchical and is related to various software process phases. This meets the other requirements for a measurement model. Use of the RADC framework in conjunction with the knowledge-base representation provides the detail needed for a computer representation of a software process model to be executed.

SOFTWARE QUALITY FRAMEWORK		FACTOR											
CRITERION	Efficiency	Integrity	Reliability	Survivability	Usability	Correctness	Maintainability	Verifiability	Expandability	Flexibility	Interoperability	Portability	Reusability
Accuracy			X										
Anomaly Management			X	X									
Autonomy				X									
Distributedness				X									
Effectiveness—Communication	X												
Effectiveness—Processing	X												
Effectiveness—Storage	X												
Operability					X								
Reconfigurability				X									
System Accessibility		X											
Training					X								
Completeness						X							
Consistency						X	X						
Traceability						X							
Visibility							X	X					
Application Independence													X
Augmentability									X				
Commonality											X		
Document Accessibility													X
Functional Overlap											X		
Functional Scope													X
Generality									X	X			X
Independence											X	X	X
System Clarity													X
System Compatibility											X		
Virtuality									X				
Modularity				X			X	X	X	X	X	X	X
Self-Descriptiveness							X	X	X	X		X	X
Simplicity			X				X	X	X	X			X

Figure 5. Software Quality Framework Factors and Associated Criteria.

The RACD quality measurements, in conjunction with explicit criteria for determining the success or failure of a quality goal, allow their use as validations for the success or failure of processes within the executing process model.

It should be noted that there are other sources of product metrics other than the RACD Quality Framework metrics. The selection of product metrics depends on the goals for a project or organization's process improvement goals and should be selected to support those goals.

2.2.5.2 Process Metrics

Venkat Ashok and Bruce Reed of UES have identified a set of metric functions and attributes for incorporation into the Knowledge-Integration Shell product <28>. These metric functions and attributes can be used to record and accumulate process information for the process tasks of specific user roles. The *KI Shell Process Metrics* represent an example of process metrics that can be used for analysis of aggregate trends. However, it is important to recognize that there are other process metrics that could be employed for analysis.

By providing a set of shell functions, the start and stop times for each measured activity can be stored in predefined information frames and attributes. From these information frames and attributes, process metrics can be displayed, both in graphical and text formats. The following metrics have been identified to parameterize the behavior of proposed KI Shell process metrics:

1. **QUEUE:** QUEUE is used to measure the time from when a work unit is created and ready for processing until the time when work is begun on that unit. Thus if queue time is greater than the acceptable aggregate threshold, the process task should be examined.
2. **WAIT:** After a role (ROLE A) has begun to process a work unit, a situation may arise where that role (ROLE A) notifies another role (ROLE B) of processing required on the current work unit. The time that the first role (ROLE A) waits on the other role (ROLE B) to complete its job is considered WAIT time. Again, excessive wait times may mean the process should be examined.
3. **PERFORM:** PERFORM represents the time that a role spends actively processing a work unit. Excessive amounts of time spent on a work unit may indicate a process problem.
4. **IDLE:** Any time a role voluntarily suspends work on a unit, e.g., lunch break, is recorded as IDLE time.
5. **SPAN:** The total time elapsed from when a role begins processing a work unit until it is completed.
6. **ITEM:** A generic metric that can be used in any manner. Associated with all the above metrics are the "actions" that can be performed on them. These actions are parameters of the metric functions and affect how the functions act upon the metrics.

For each of the above metrics there is a set of predefined attributes used to store the start, stop, and accumulated times. The action parameters determine which of and how the attributes are set. These action parameters are:

1. **START:** Sets the start attribute of the specified metric to the current time (in seconds).
2. **STOP:** Sets the stop attribute of the specified metric to the current time (in seconds).
3. **ACCUM:** Calculates the difference in seconds between the start and stop times for the specified metric and adds that value to the current value of the "accum" attribute of the specified metric.

From the analysis of selected process metrics for each process or set of process tasks, valuable data on the execution of the processes themselves can be used to identify trends, both positive and negative, that can be used to support process improvement.

Intentionally left blank.

3.0 STARS IS-15 Task Organization

IBM STARS Task IS-15 was organized into three major tasks to satisfy our process objectives. These three tasks were:

1. Software Process Representation
2. Software Process Enactment - Cleanroom Software Process Case Study
3. Process Products Coexistence Strategy.

The "Software Process Representation" task comprised three subtasks:

1. SPMS Evaluation Prototype Demonstration and Training - to demonstrate the SPMS evaluation prototype, to gain interest and commercial support for cofunding during the "T" increment, and to prepare materials for training the "SEI/STARS Process Asset Acquisition" group in the use of the SPMS Evaluation Prototype.
2. SPMS RISC System/6000 Port Analysis and Planning - to assess the feasibility of porting the Apple Macintosh-based SPMS prototype onto the IBM STARS SEE platform, namely the IBM RISC System/6000 running under AIX, and to prepare a plan for doing so.
3. Process Representation Using Box Structures - to examine the use of box structure notation as a candidate notation for recording process knowledge.

The Software Process Enactment - Cleanroom Software Process Case Study comprised two subtasks:

1. Case Study Specification and Validation - to define the *"Cleanroom Software Process Case Study" problem*, to specify a concept of operations for a system to support the *Cleanroom Engineering Software Development Process* (the *"Cleanroom Engineering Process Assistant" Specification*) and validate the implemented prototype against the process and specification prepared.
2. Case Study Implementation - to implement the *"Cleanroom Software Process Case Study" Problem* as required by the *"Cleanroom Engineering Prototype Assistant Specification."*

The Process Products Coexistence Strategy Task involved examining how selected products -- KI Shell, HP SoftBench and SPMS, each of which provides a needed software process management capability -- could be integrated to provide a unique software process management capability.

Figure 6 on page 26 illustrates the IBM STARS "S" Increment Process Task Team and identifies all participating team members.

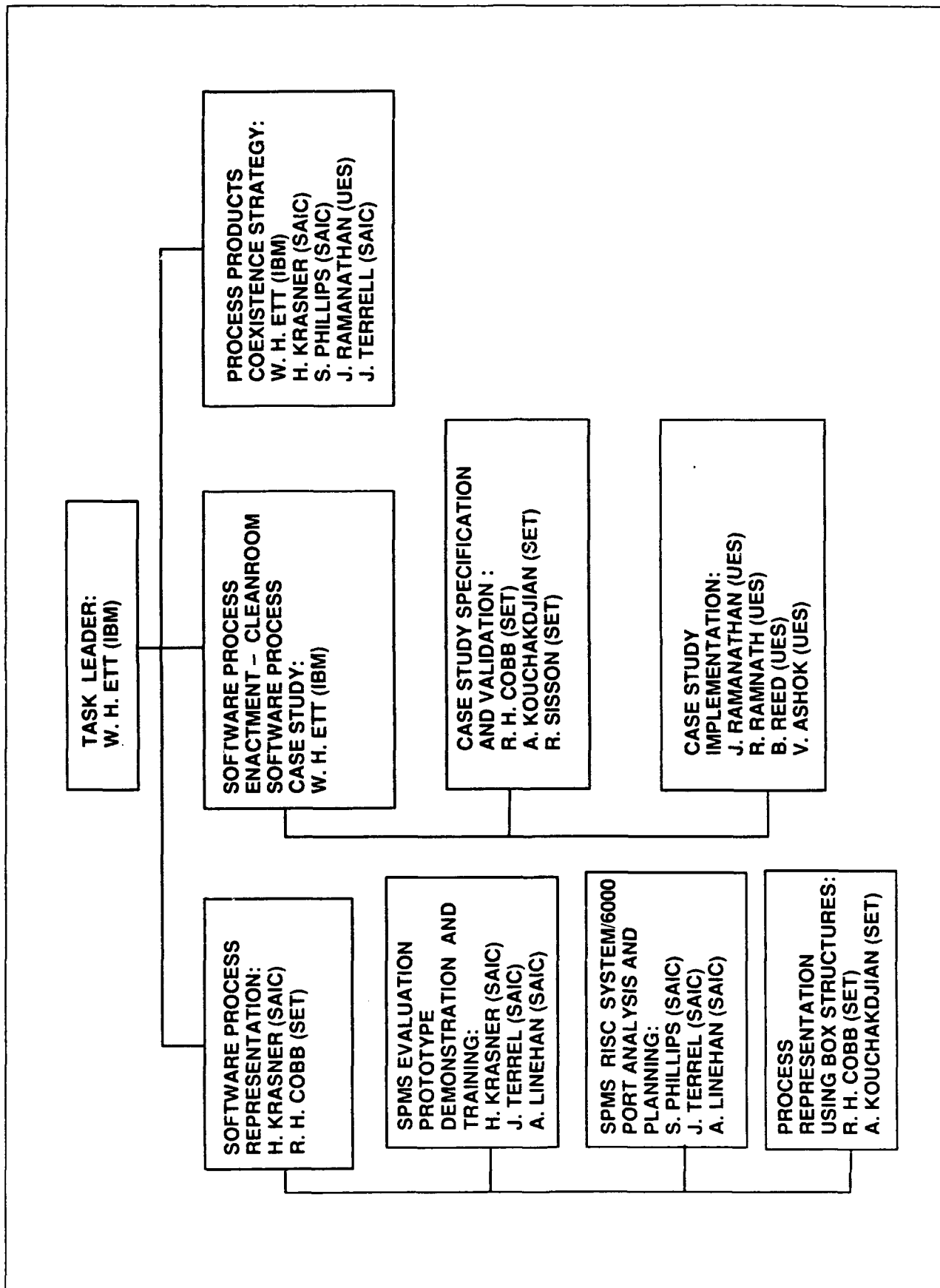


Figure 6. IBM STARS "S" Increment Process Task Team.

4.0 STARS IS-15 Candidate Tool Acquisition

This section describes the tools selected to support STARS Task IS-15 and describes constraints on the selection of tools, as well as the rationale for tool selection.

4.1 Constraints on Tool Selection

1. IBM examined several candidate tools to supporting software process modeling and enactment.
2. Constraint criteria on tool selection:
 - a. Availability to support "S" increment work
 - b. Potential availability of tool for the 1993 STARS SEE
 - c. Product available currently as commercial off-the-shelf (COTS) or as a stable prototype
 - d. Potential for migration to multiple platforms supporting POSIX.

4.2 Tool Selection for Providing a Software Process Modeling Capability

IBM has selected the Software Process Management System (SPMS) as one of its candidates to support software process modeling. SPMS is one of the few tools designed to support the concept of modeling software process and is why IBM has decided to examine how to migrate SPMS to its RISC System/6000 platform.

Existing CASE tools that can be used to model aspects of process, support process modeling in a non-integrated fashion. Although some CASE tools do support the simulation execution of systems, their usefulness as tools for process engineers is somewhat limited. System simulation capabilities for tools such as STATEMATE and TEAMWORK/SIM are intended to simulate the execution of system designs. Although process models can correctly be viewed as process systems, the simulation of process models requires a different "look and feel" than do other system designs in that humans, such as program managers and engineers, have to be modeled as process agents (enactors of process). SPMS was designed to be a tool for process engineers to model and simulate processes. Further SPMS was an attractive candidate for the following reasons:

1. SPMS's availability to support "S" increment work

The SPMS prototype is available to support process modeling experiments on an Apple Macintosh platform. Further, it serves as an excellent "requirements" prototype for migrating SPMS to the IBM RISC System/6000.

2. The ability to make SPMS available for the 1993 STARS SEE

The Software Process Modeling System was the most mature software process modeling capability that could potentially meet the objectives of supporting reuse-based process system modeling.

The SPMS prototype was designed to take advantage of popular commercial-off-the-shelf tools that can be found on a number of UNIX workstation platforms including the IBM RISC System 6000 under AIX and SUN workstations under BSD 4.6. These tools include Oracle and NEXPERT Object.

3. SPMS is currently available as a stable prototype, which has the potential to become a commercial off-the-shelf product.

SPMS is currently a stable prototype. The SEI is planning to evaluate SPMS for use in support of their process modeling and asset capture activities.

4. Potential for migration to multiple platforms supporting POSIX.

SPMS was designed around integrating COTS products that are generally available on POSIX-compliant UNIX platforms. SPMS will be developed employing industry standards, such as X-Windows and OSF/Motif and the COTS products selected. Application integration toolkits, such as the services provided by HP SoftBench, may differ from platform to platform, and represent the area where SPMS customization for each unique platform will be required.

The decision was made to begin assessing what it would take to migrate the existing SPMS capability to the IBM RISC System/6000 to provide the IBM SEE with a tool to support process modeling and process enactment. In summary, this decision was based on the fact that SPMS has already demonstrated its ability to support process modeling and process simulation, and SPMS holds the potential to support process monitoring.

For further information on the Software Process Management System, please refer to the following STARS reports:

1. "A Software Process Management System for the STARS Software First Life Cycle," IBM STARS Deliverable CDRL Number 3016, 29 October 1990.
2. "User's Manual for SPMS," IBM STARS Deliverable CDRL Number 3118, 17 June 1991.

4.3 Tool Selection for Providing a Software Process Enactment Capability

IBM selected *KI Shell (Knowledge Integration Shell)* to support STARS software process enactment experiments. KI Shell is one of the more mature software process enactment capabilities available as a commercial product. It is available for immediate use on an IBM RISC System/6000. Further, UES, the developer of KI Shell, is consulting with IBM AIX-CASE planners on software process management capabilities desired for future releases of IBM's AIX-CASE.

1. Availability to support "S" increment work

KI Shell is the only capability for performing process enactment experiments on an IBM RISC System/6000 platform. Because KI Shell has several years of testing behind it, IBM has selected it as its candidate software process enactment tool to support its STARS "S" increment work and at the same time, to evaluate its potential for supporting the 1993 STARS SEE.

2. Potential availability of tool for the 1993 STARS SEE

KI Shell is a commercially available product that has had significant use in implementing manufacturing processes and some experience in supporting software processes.

Work is currently going on to examine making KI Shell, PCTE-compliant to work with existing PCTE-based SEE frameworks.

3. Product available currently as COTS or a stable prototype

KI Shell is commercially available. Further, based on the "Cleanroom Software Process Case Study" implementation work, several ideas have emerged to improve KI Shell's ability to support software process enactment.

KI Shell has been used successfully to implement a variety of manufacturing processes. Background information about KI Shell will be provided in the next section.

4. Potential for migration to multiple platforms supporting POSIX.

KI Shell is a commercially-available product for the IBM RISC System/6000 running under AIX. It is also available of the following platforms:

- HP 9000 under HP/UX
- SUN 3 under SUN/OS.

Further, porting the KI Shell to other UNIX environments can be accomplished in a one- to three-month time frame. This is based on the port of KI Shell from the IBM PC/RT AIX implementation to the following UNIX versions:

- HP/UX - 2 person months
- SUN/OS - 3 person months
- AIX - 1 person week.

Intentionally left blank.

5.0 The Knowledge-Based Integration Shell

The KI Shell (Knowledge-Based Integration Shell) is a commercially available process execution and control shell. This shell reflects the process-based integration concepts and was commercialized using Phase II SBIR funding from the Air Force³ as well as other sources. The KI Shell process definition and enactment mechanisms have been tested and can be viewed at the McDonnell Douglas's *Team Columbus* production site where KI Shell is used for improving the productivity of complex manufacturing processes involving multiple roles. At *Team Columbus*, the use of KI Shell demonstrates how the STARS vision of network-based collaborative development can be achieved.

Under the "S" Increment of the STARS Program the IBM STARS S-15 Case Study Implementation Team employed KI Shell to support process modeling and enactment. As determined in the IS-15 increment, the KI Shell currently does not have a direct competitor in the IBM/AIX market. Further, during the "S" increment, the feasibility of using the KI Shell to model and enact processes, such as the Cleanroom Process, has been demonstrated by the IBM STARS Team.

The UES staff on the IBM STARS S-15 Case Study Implementation Team has a twelve-year history of research and development in problem areas directly or indirectly related to those of STARS. UES has developed, under contract to the USAF ManTech Program, and commercialized a unique object-based meta system -- the KI Shell -- which can take any process and method description and enact it to provide active support for any process, including software engineering processes.

5.1 KI Shell View of Process Technology

Process related technologies include process modeling, process simulation, process management, and process enactment. The KI Shell features support Process Modeling, Process Enactment, and some aspects of Process Management. These features have been developed and validated through extensive application in concurrent, collaborative engineering.

Process-related technologies for collaborative work flow must provide features for at least four related aspects of business/technical work-flow processes based on the organization:

- Generic Process Modeling to acquire and specify the process knowledge necessary to perform the enterprise activities. This includes the ability to specify enterprise roles and responsibilities (activities that must be completed by a role), relationships between roles and the external views of functions and information necessary to perform individual activities.
- Process Simulation to determine precisely how different activities in a specific model consume resources over time and, thus, to identify bottlenecks and areas that need attention for continuous process improvement.
- Process Enactment to provide decision support and to assist in completing each activity in a model by providing information on a need-to-know basis to make decisions, using operations to determine the external view, and recording key decisions.
- Process Management to provide a management view of the actual status of decisions with respect to each activity in the work-flow process by which product information is produced. Project Management is also an aspect of this, as it is necessary to plan or replan the use of resources for different activities.

³ UES contract under Phase I SBIR.

The KI Shell Development Environment allows the modeling of a method. A KI Shell *Method* is a model or description of *roles*, *activities* that constitute the work-flow process that must be completed by each role, *applications* (or implementation systems) that must be invoked within activities, and *data* that must be manipulated. In the KI Shell an object-based approach is used to record and enact a method. As the method is enacted, precise global process status data are available for management viewing.

The example in Figure 7 is an overview of a method description comprising activity objects, use of enterprise data, use of applications, and constraints/controls affecting the decision at each activity. This method example, developed for the Air Force, supports the die design process for extruding complex alloys, by employing concurrent engineering practices. More specifically, the first activity is "Product Specification." When completed, it provides the *Geometry*, *Application*, and *Microstructure* values for constraining decisions made at later activities. For example, the *Geometry* constrains the candidate billets selected from a manufacturing database during the "Billet Selection" activity.

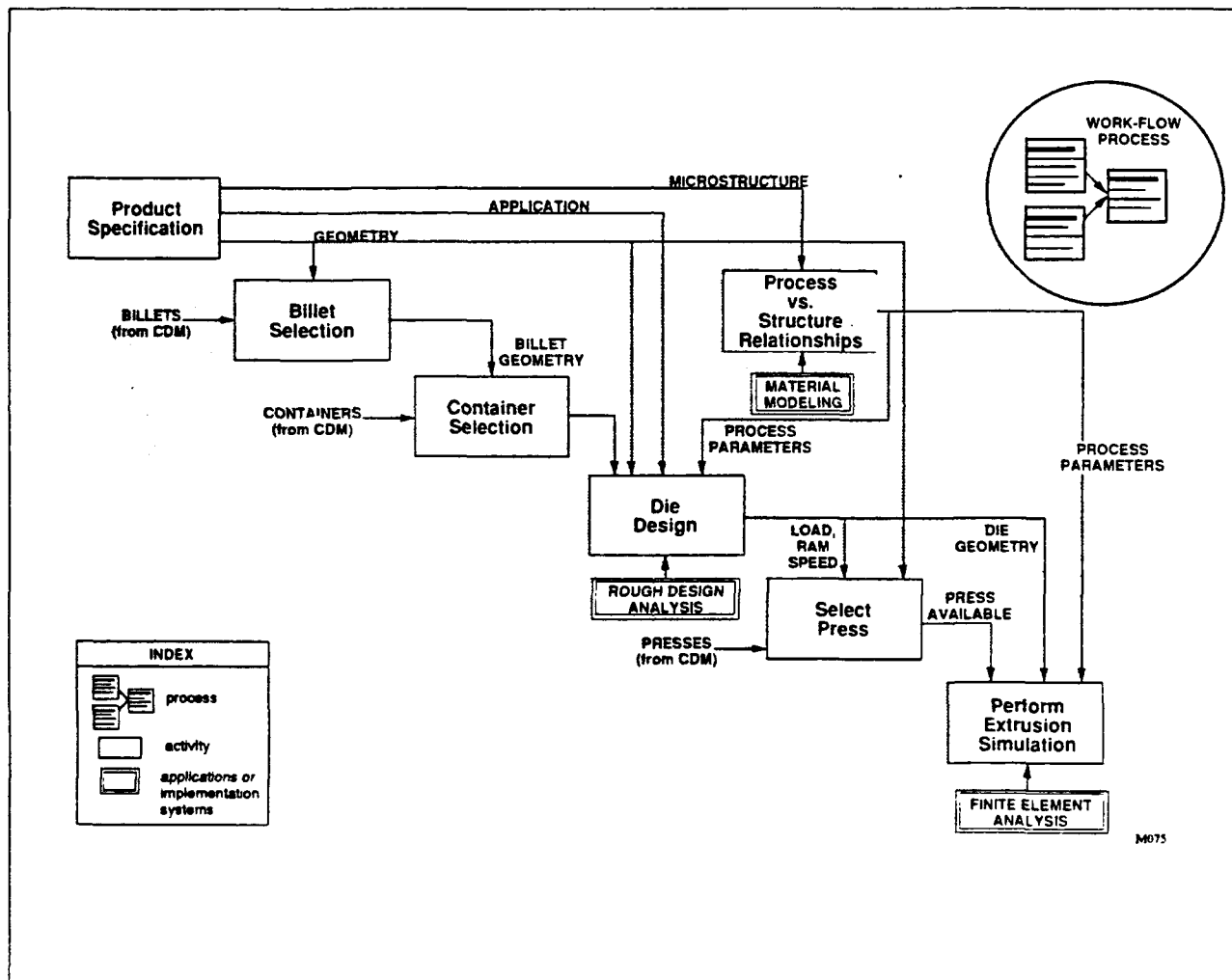


Figure 7. Example of a Concurrent Engineering Process Performed by a Die Designer Role.

The actual geometry of the billet *selected* by the engineer becomes the decision value of the attribute associated with enactment of "Billet Selection" activity. This value, in turn, constrains the next activity and so on. "Die Design" is an example of an activity that invokes an application for rough design before detailed finite element simulation during the "Perform Extrusion Simulation" activity. This activity, in turn, is performed with the assurance that the needed press, necessary for guaranteeing the appropriate processing conditions, is available.

Figure 8 on page 33 illustrates how an activity in a process, such as the example in Figure 7, is enacted. When an activity is performed or enacted, a procedure associated with the activity is executed by the KI Shell execution monitor. This procedure typically will use enterprise data, examine earlier decisions made when other activities were performed to assist in the current decision, invoke applications on the right data, update appropriate attributes of activities and databases, and finally, create process instances for roles as necessary. This is illustrated in Figure 11 on page 36.

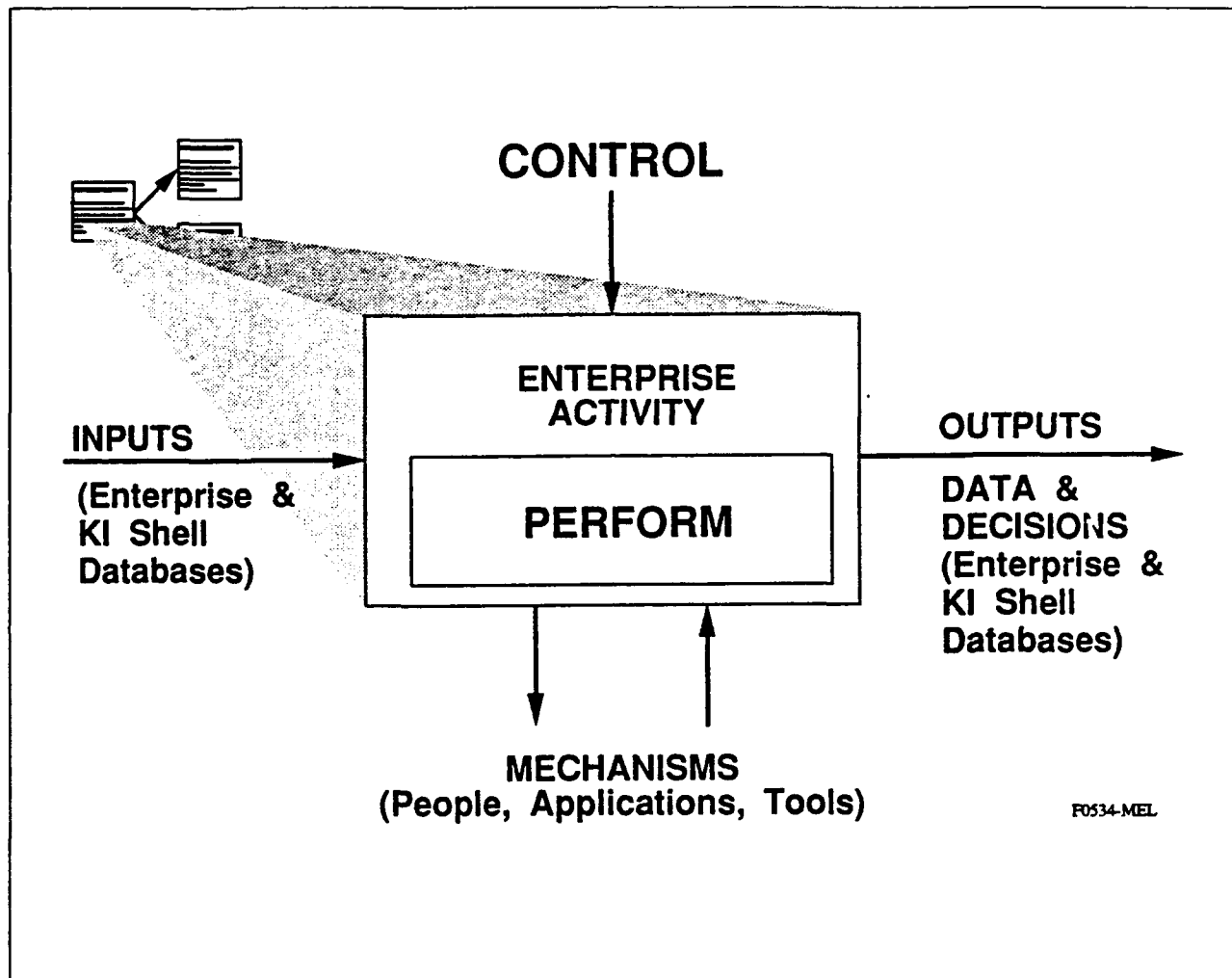


Figure 8. Enterprise Activity Must Be Performed By Using Information / Mechanisms under Appropriate Control.

Activities also can be structured as processes to be performed by different roles in an organization according to the responsibilities. Coordination between different roles, to ensure activities are performed correctly, is also supported by the KI Shell. Figure 10 on page 36 illustrates a process for creating and reviewing information. In this example the subprocess instances for roles are dynamically created over time.

"TO-BE" Method *design* requires interaction with end-users to determine the requirements of the Computer-Aided Manufacturing (CIM) system. Figure 7 on page 32 is an example of a concurrent engineering process performed by a single die designer role.

To use the KI Shell, a method must be first designed for a specific enterprise function. IDEF0⁴ modeling techniques can be used to define the "AS-IS"⁵ and "TO-BE"⁶ processes. Once a method is agreed upon, a machine-readable form of the method is rapidly created by the *Method Designer*, using the KI Shell *development environment*. (See Figure 9 on page 34.) This method description is stored in an SQL database. The machine readable form of the "TO-BE" method is enacted by the KI Shell execution environment and the end-users are guided according to the method.

XTRUDER, funded by the Air Force and developed by UES, demonstrates that process execution can reduce die design time from months to a few days. Dramatic productivity gains have also been demonstrated at GM Allison. Here the work-flow process involves collaborative design of turbine blades by different roles - stress, aerodynamics, mechanical, etc.

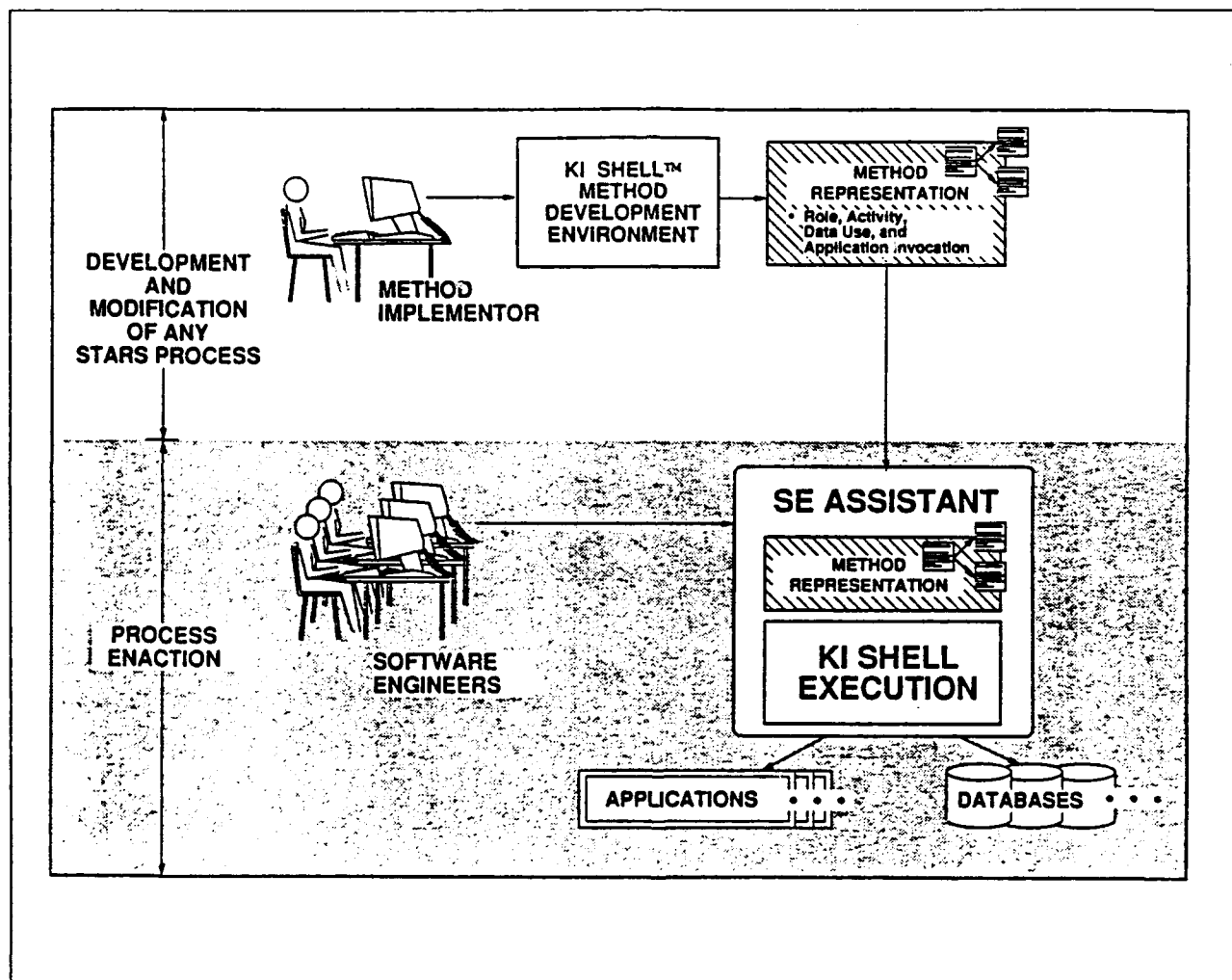


Figure 9. The Development and Use of KI Shell Method.

⁴ IDEF0 is one of the modeling tools of the IDEF (ICAM Definition Method) methodology. IDEF0 is referred to as the function model which provides a description of a manufacturing system in terms of a hierarchy of functions. The basic tool of IDEF0 is the activity diagram which illustrates data flow between functions, control and stimuli for a function, and the mechanisms employed by a function.

⁵ "AS-IS" process refers to a system of processes that currently exist to support the development or production of some product, e.g. a software system, a computer system, etc.

⁶ "TO-BE" process refers to the desired system of processes needed to support the development or production of some product.

The term *Assistant* is used to describe the final software comprising the KI Shell method, the KI Shell runtime utilities that interpret and enact the method, and the interfaces to the applications. When an Assistant is used by the user of the CIM system, the method guides the decision-making during the process (collection of activities). For the current activity, the KI Shell execution monitor invokes the perform procedures that control the invocation of applications and the use of data.

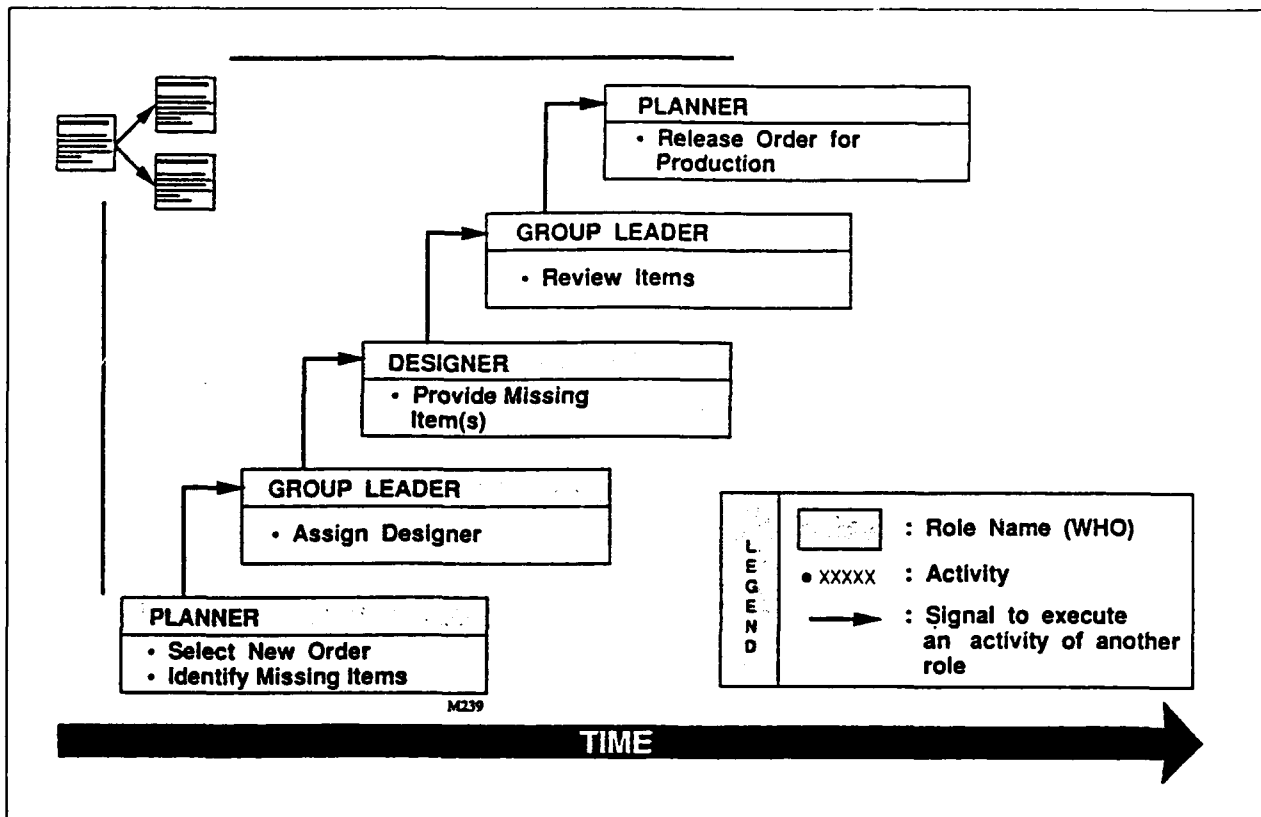


Figure 10. ACTIVITY & ROLES: To Support a Generic Enterprise Sub-Process.

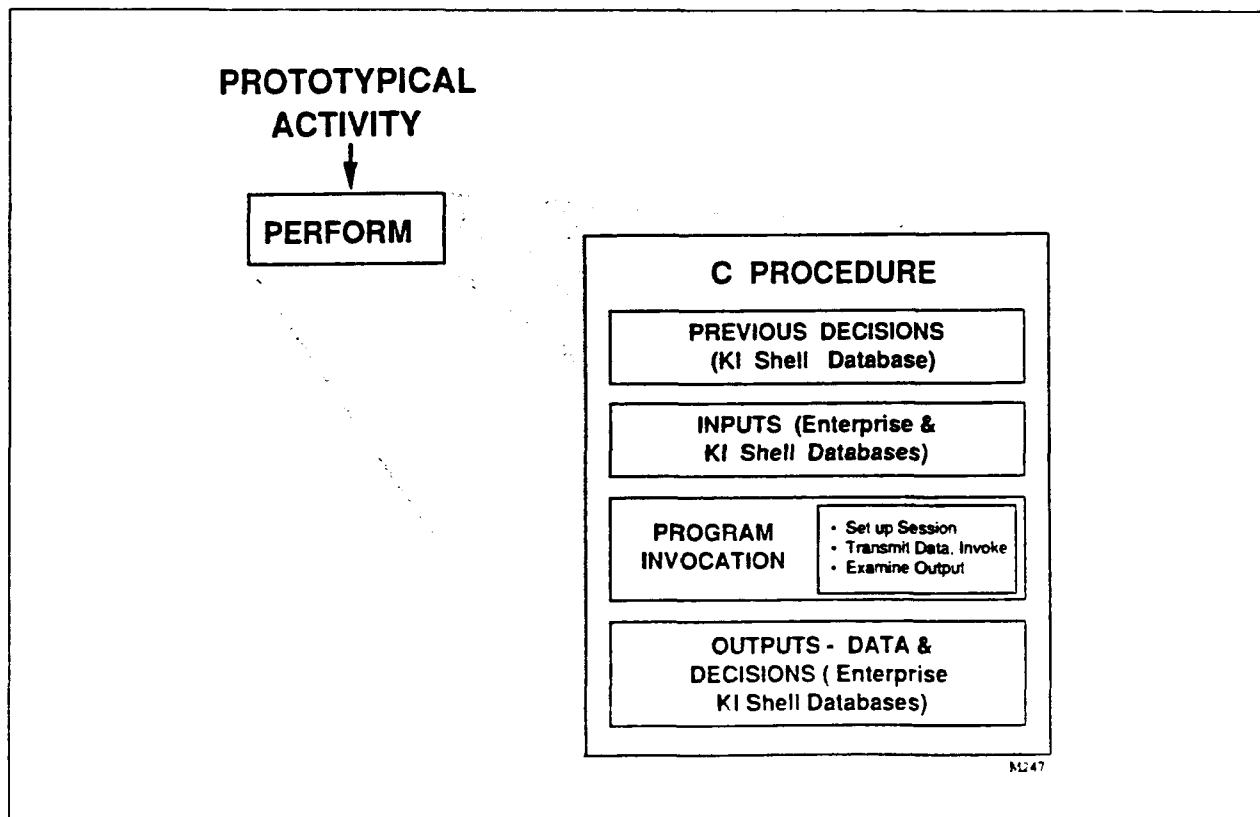


Figure 11. PERFORM EVENT CAUSED BY A MOUSE CLICK AT THE USER INTERFACE: Causes Execution of a C Procedure.

5.2 Summary of KI Shell Features

The key features for modeling processes in KI Shell are:

- Modeling of roles and the structured process to be completed by a role with no programming effort required.
- Ease of development of procedures to be enacted.
- A library of reusable code available for programming productivity.

The key features for supporting process enactment in KI Shell are:

- Enaction of procedures including invocation of applications on the correct data to complete activities in the process.
- Guidance for the end-user based on the current activity.
- Maintenance of global process state.

The key benefits provided by KI Shell to support the development of process systems include:

- Knowledge of work-flow process control is made explicit and thus easier to model, customize, and evolve.
- Reduced assistant development time, with significantly fewer lines of code are required to develop KI Shell process assistant (system to support process enactment).
- Layered and incremental assistant development with more done by domain expert and less done by the programmer.

5.3 KI Shell Concepts

The term *Method* refers to a collaborative process enacted by different roles, with each role executing a sub-process using data and tools. In this section we present the Method meta language (a language for defining methods) related to the unique process enaction technology developed by UES scientists over a twelve-year period. This meta language is critical to supporting process requirements for STARS, because it enables complex processes to be described and executed. The Method meta language facilitates easier modification and evolution of any process.

5.3.1 Method Meta Language

The term *Method* is used here to refer to a specific software engineering method. On the basis of our long-term software engineering research efforts to develop a methodology or Method meta language⁷, we have concluded that any software engineering method can be decomposed as a process consisting of activities (or verbs), data objects (or nouns) created and modified during that process, tool fragments used within activities, and the roles responsible for performing the activities. More generally, a method is a discipline ("when") by which product data objects ("what") are created and changed to different states by specific individuals ("who") using specific procedures ("how"). Some of the key terms of the Method meta language are:

- Activity:

⁷ By language we mean the collection of syntactic and semantic conventions and the collection of System Integration Library functions that are available via the KI Shell Development Environment. The interface presented is more like an environment as opposed to having to write statements in FORTRAN.

A basic unit of work in terms of "what" and "how". An activity is a slot with an associated completion condition. The "how" is described in a C Rule or Procedure with SIL (KI Shell System Integration Library) calls.

- **Attribute:**

Placeholder for data values in a process frame. A variety of complex attributes like sets, arrays, and text are supported.

- **Complete:**

Status of an activity that allows users to progress to the next activity.

- **Data/Information:**

Data used or created when performing activities. The data can be stored in diverse sources such as in external databases, in files, or in attributes of KI Shell process and information objects.

- **Method:**

Body of knowledge, policies, and procedures that can be created and maintained by multiple roles. Specifically, a method consists of process frames grouped by roles.

- **Method Instance:**

History or audit trail of the enactment of the process/method for a specific product.

- **Perform, Premodify, Postmodify:**

Examples of events that cause rules (triggers) to fire.

- **Process Frame:**

Aggregation of a set of activities structured by a control construct that determines the sequencing of activities. The order of activities can be "sequential," "choice," "if-then," etc.

- **Process Frame Instance:**

Instance of a process applied to a specific product.

- **Programming Interface (PI):**

A C callable program that invokes the tool on specific data and returns specific values.

- **Role:**

A collection of activities performed by a prototypical project member. A role is a network of process frames. A user can execute different roles.

- **Rules:**

KI Shell/C Procedures (triggers) associated with activities that assist users in performing activities, propagating values, implementing constraints, etc. The body of the program can have calls to SIL that manipulate the process and information frames. Rules used for computation, monitoring, notification, tool invocation, and transformation.

- **Sequential, Choice, If-Else, While:**

Attributes of a process frame that determines the sequencing of activities within the frame.

- **Slot:**

A group of attributes, rules, and links.

- **Subactivity:**

The refinement of an activity by another process frame. Completion of the subactivity process implies the completion of the parent activity.

- **Tools:**

External Programs invoked to perform an activity.

- **Wait and Send Signal:**

Synchronization primitive, which suspends activity until it receives a signal from other activities in other roles.

- **Information Frame Features:**

A collection of slots with the following properties: object-valued attributes (one-way), links (Two-way), and complex attributes (sets, arrays, text). Rules are triggered when data is modified and the action part of rules can access frames database, and interact with the user, invoke tools, etc.

Figure 12 on page 40 is an example of a more detailed method layout for a specific software engineering process based on the waterfall process model. This method, when enacted by the KI Shell, will coordinate between roles. For example, in the sequential frame for the "Projects Manager" role the user cannot proceed with the "Approve Systems Analysis" activity until the "Systems Analysis" role executes to completion.

5.3.2 KI Shell Process Modeling and Enactment Features

Features for supporting KI Shell method development (process model development):

- Includes a Method Development component, based on the KI Shell Method meta language, that provides a declarative environment to create and modify complex methods.
- Enables rapid and incremental development and modification of a method for any SE processes.
- Permits method objects to be stored in SQL. The Method Development Environment has a schema editor that allows Method Objects to be maintained in a database. A Programming Interface (PI) to method objects is available.
- Provides a productivity tool to generate the body of rules associated with the methods. Object-based query functions that can be embedded in procedural components to query process and information objects are also available.
- Includes the System Integration Library (SIL) functions, productivity tools, and object-based query functions reduce the lines of code to be written by the developer of a specific SEE (Software Engineering Environment).

Features for supporting KI Shell method enactment (process enactment):

- Permits rule invocation (triggers) to perform the "How" on the basis of events related to the current activity, role instance, and information objects.
- Executes process and invokes program interfaces (PI) to tools on the basis of the right data in the context of executing activity models.
- Maintains an audit trail of the process execution in an underlying SQL database.
- Coordinates between multiple roles on different workstations (one role can create processes to be executed by another role).
- Allows a user to execute different roles and instances in different windows.
- Provides a multi-window, graphical, and iconic user interface using industry-standard OSF Motif to present executing processes iconically.
- Permits a LAN configuration with a process server database, where different role instances can be executed on different workstations.

CaseMgr™

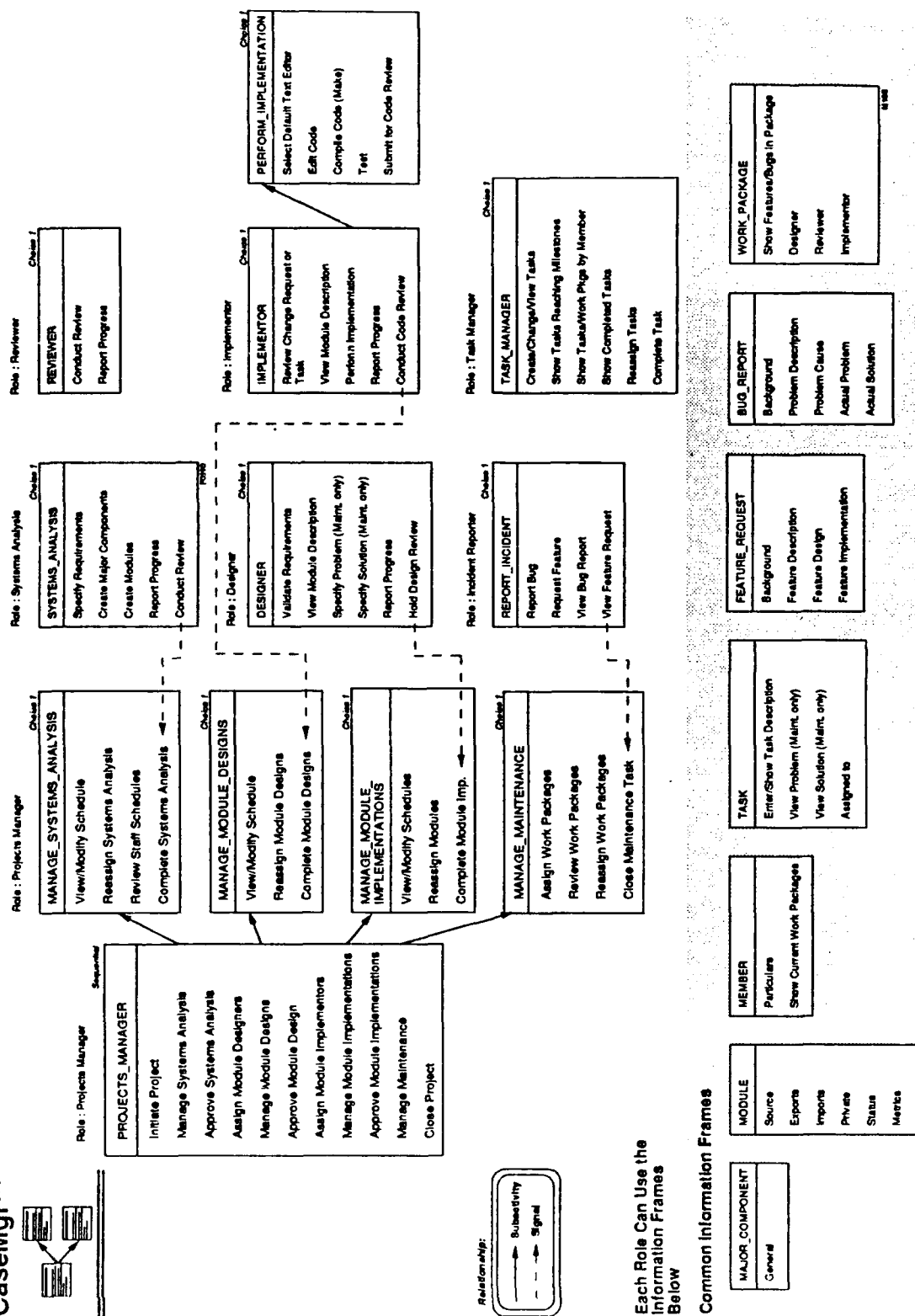
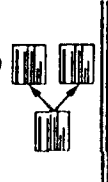


Figure 12. Method Layout for UFS's CASE Manager.

KI Shell (Figure 13 on page 42) has been engineered over years as an open-architecture system written in C. A process execution shell must orchestrate interactions between end-users, tools, and data objects according to a specific method. To enforce such a discipline, the execution monitor must have detailed control over the activities of the end-users as they invoke tools, perform activities, and modify data objects. The detailed control is achieved by building the shell as an intermediary (executive) program among the user, the tools, and the shell's own object-based database (Figure 14 on page 43). By having this control, the process model can enforce a discipline by utilizing its represented knowledge. Finally, the process types and instances must be persistent. By saving the process instances -- the decisions made at each activity -- an audit trail of the enterprise work-flow process can be maintained. Such a persistent store must also provide the usual database features - multiuser, crash recovery, etc. The KI Shell has been engineered over years to meet these requirements.

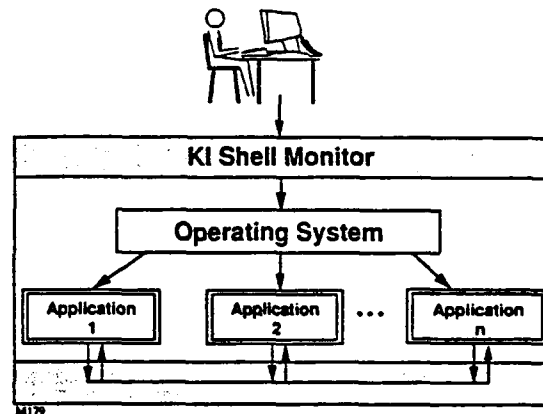
5.4 Installation of a KI Shell Application

The KI Shell is implemented in C⁸ and runs under AIX, UNIX, VMS and DOS. The method descriptions are stored in an SQL server database on the workstation. A SE Assistant of the KI Shell is a method with tool invocation program interfaces to existing software (for example, design, analysis, etc.). In a typical customer environment, the existing software systems may run on heterogeneous hardware. Thus, a program interface to each of these tools is required. To support the method in a local area network with roles executing on different workstations, one of the workstations must have an SQL server database and the other workstations must run the database client software.

⁸ Early prototypes include a LISP (KRL)-based system completed in 1982, EMACS UNIX-based system completed in 1984, IBM VM based system completed in 1986, IBM VM-based product with an underlying database in 1988, VAX VMS-based system in 1988, and the current product version using C and SQL in October 1989.

KI SHELL Monitors And Mediates Between:

- User And Application
- Application And Application
- User/Application And Database
- No Performance Penalty At Execution Time



Traditional:

- User Invokes Each Application

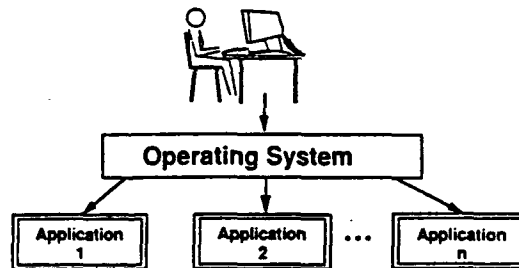


Figure 13. KI Shell's Runtime Architecture.

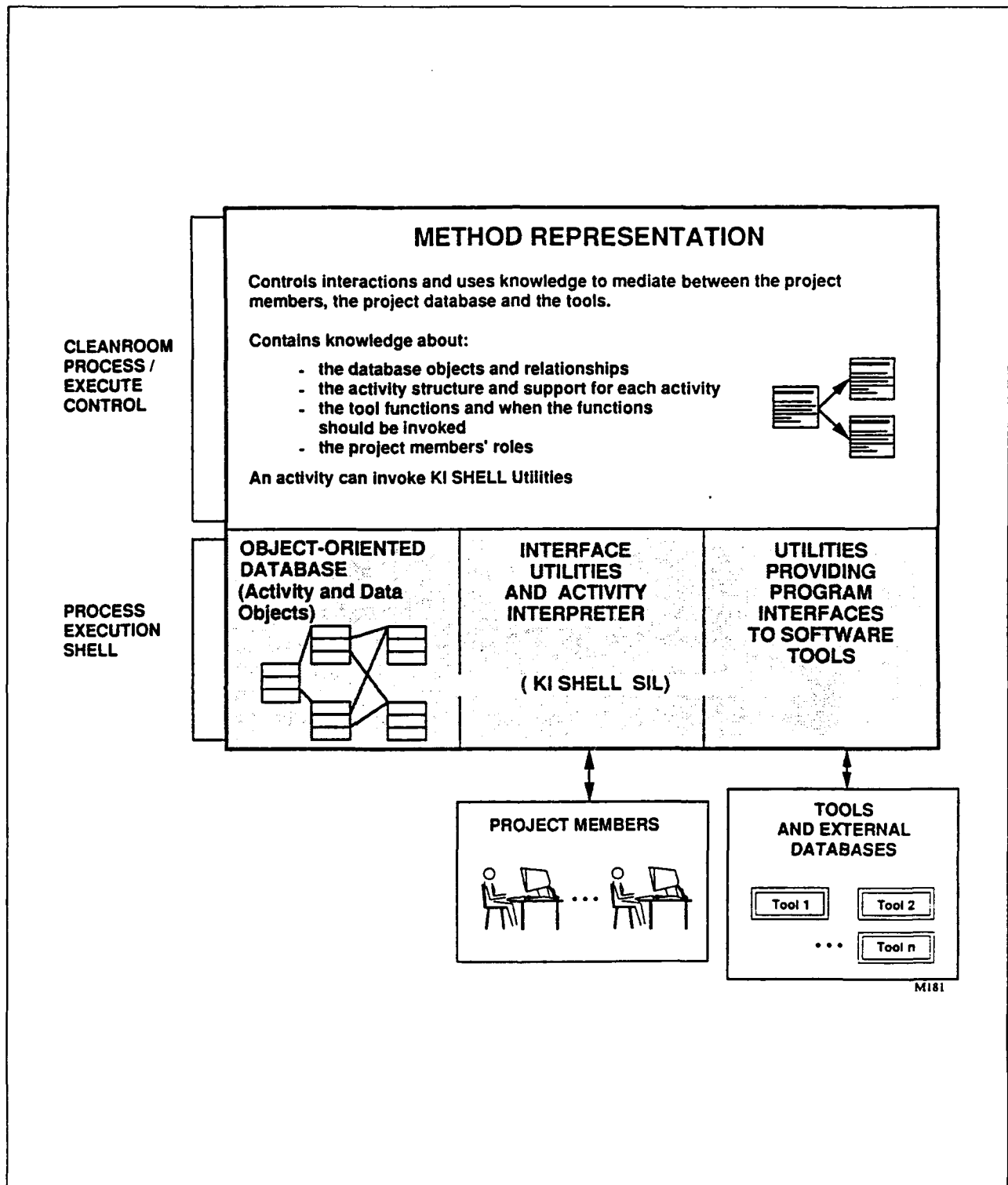


Figure 14. Knowledge-Based Shell Will Be Used to Implement a Specific KI Shell Application. Applications Include "Software Engineering Assistants."

Intentionally left blank.

6.0 STARS IS-15 Software Process Case Study Preparation

6.1 Preparation and Scoping of "Cleanroom Software Process Case Study"

This section briefly describes the activity and records the lessons learned during the creation of "The Cleanroom Software Process Case Study" deliverable.

6.1.1 Brief Description

This task was completed in two steps. The first step was to understand the goals and strategy for the case study. Upon its completion, the second step was to prepare the case study problem document.

Initially, a number of group discussions were held by the specification team to clearly understand the purpose of a "Cleanroom Engineering Process Assistant (CEPA)" and the purpose of the case study. The more general questions of what is a process, what is process modeling, what is the purpose of a process assistant, and how do humans interact with a process were discussed. In understanding the general concepts, specific insights were gained. These discussions helped determine the size and scope of the case study and the strategy for the demonstration. After each discussion, the present state of understanding was noted, along with open questions and action items. This allowed the body of knowledge and understanding to increase, while the specification team became more focused on the end result of the discussions, which would be the writing of the case study. Additional discussions helped focus the case study-specific concepts more closely. The size and scope of the case study was determined. The parts of the Cleanroom process that were to be used for the case study needed to be carefully listed and described. Additionally, the scope of the case study, a description of what the Cleanroom process was going to be applied to, was also determined. This was the "Host-at-Sea Buoy" problem < 23 >. Both the size and scope needed to be well thought out, since time allotted to perform the case study was limited to three staff months.

The deliverable was written by first determining the goals of the case study, and then supporting the goals with additional information necessary to make the case study description complete. A sequence of versions was created, each of which was reviewed, with the comments being used to improve the next version of the document. After a number of revisions, the document was rewritten to improve clarity and to improve organization. The document first presented an overview, which described the purpose, approach, assumptions, and relevance of the case study. Subsequent sections described the case study in a more detailed manner (including discussions of the Cleanroom process, CEPA, and the "Host-At-Sea Buoy" problem), the test bed and scenario descriptions, a proposed demonstration scenario, and evaluation questions (for both the Cleanroom process and CEPA). Additionally, the "Host-At-Sea Buoy" specifications were included, as was a discussion of the relevance of the case study to process modeling work previously done by Kellner and Rombach for the 6th International Software Process Workshop < 14 >.

6.1.2 Lessons Learned

The specification team could only select a portion of the Cleanroom process model, given the project's time constraints. To capture the entire Cleanroom process would have been a significantly larger task.

Ensuring that the portion of Cleanroom selected for CEPA contained all of the criteria that Kellner and Rombach described in their paper "Comparisons of Software Process Descriptions" was a useful exercise. The criteria identified helped the specification team make the portion of the Cleanroom model selected and the buoy problem more complete. The Kellner/Rombach criteria gave the specification team a checklist of issues to consider when developing the case study.

Although it was attempted for the "Cleanroom Software Process Case Study" problem document, a script for a test or demonstration scenario is difficult to create until the prototype is complete. Although specifications can be prepared for a prototype concept, there are no guarantees that the specification for the prototype system will be completely and correctly implemented. Additionally, it is necessary to work with the final prototype product to better understand how to demonstrate the concepts built into the prototype system.

Writing an implementation-free specification of the buoy problem was a straightforward exercise using Mill's *Box Structure* notation <24>. Further, the *Construction Plan* was also relatively easy to prepare using "Box Structure" notation. This was somewhat surprising since the "Host-at-Sea Buoy" problem is not a trivial application.

6.2 Preparation of Specification for the CEPA Demonstration

This section briefly describes the activity and records the lessons learned during the creation of the *"The Cleanroom Engineering Process Assistant (CEPA) - Specifications for a Prototype of the Workstation Component and Dispatcher."*

6.2.1 Brief Description

The primary goal of the specification process was to create a specification for the implementation team to use, to develop a prototype *Cleanroom Engineering Process Assistant* capable of supporting the enactment of the *Cleanroom Engineering Software Development Process*. Since this is a case study, an objective was to observe the specification creation process to gain experience for an eventual production version of the CEPA system.

The perspective from which the specification and validation team tried to view CEPA was that of a process engineer who is trying to develop an automated support application for a project's process. From that perspective, the specification and validation team needed to understand the underlying process and to identify the parts of the process in need of implementation. The specification was created by first spending a significant portion of time to understand the problem and solution domain. Specifically, the specification and validation team's focus was on understanding desired capabilities for CEPA in specific and for a process assistant in general. The specification and validation team tried to understand what an ideal CEPA would be like. The team took advantage of experiences individuals have had creating and using other systems, both inside and outside the CASE domain. Looking at a potential system from an abstract perspective allowed the specification and validation team to understand the problem as completely as possible before embarking upon a specification of the solution. Of course, some "problem domain" analysis occurred while the specification was being developed. The specification was then created, with the intent of clearly specifying all functionality that was necessary, without imposing a specific solution since that was the responsibility of the implementation team. Five versions of the specification were written. Most of the specification remained relatively constant after May 13, which was the second version. A portion of the time writing the specifications consisted of discussions with the implementation team, to explain the specifications and to assist the implementation team in developing the prototype. This was done since the specification and validation team authored the specifications and was the resident expert on the Cleanroom process. In terms of an effort distribution for the specification, approximately 45% of the specification and validation team's time was spent understanding the problem and solution domain, 45% was expended in writing the specification, and 10% was shared in working with the implementation team.

The final version of the CEPA specification describes the set of operations that CEPA can perform (from a user's perspective), that will support the software development project members in conducting a Cleanroom project. A number of different roles that staff members can perform are described, as is a significant amount of functionality necessary for the Cleanroom process. The specification did not present a complete description of the Cleanroom process, nor did it describe all of the activities for all staff members related to a project. The specifications were purposely limited by the size of the case study. Primarily the

development/certification cycle of the Cleanroom process was specified, since the purpose of CEPA was to support the case study.

6.2.2 Lessons Learned

Working on the *User's Manual* portion of the specification was important, since the possible stimuli for each user were clearly defined in that section. Additionally, the *User's Manual* is the primary section where a developer can get an idea of the intended 'look and feel' of the software, although the *User's Manual* should not dictate a specific implementation.

The use of the notation for black boxes, as defined by the *Box Structures* notation <24>, for the specification and design of systems allowed for an implementation-free description of CEPA, where only the responses in terms of stimuli histories were presented.

Writing a box structure *Black-Box Specification* seemed to be extremely efficient. The specification and validation team is not sure that a similarly good specification could have been written, given the same time constraints.

Transaction descriptions were used to show sets of related stimuli, to give an implementer a better idea of the CEPA transaction usage, e.g., a sequence of stimuli. The transaction descriptions also helped the specifier to ensure that all stimuli were used, as well as used correctly.

Use of abstractions helped make the description of the box structure black-box subfunctions more compact.

Relatively complex systems can be described quite compactly when a box structure black box description is used. Less than 100 stimuli were required to describe CEPA.

Looking at CEPA from multiple user's perspectives (the different human users, as well as the dispatcher and storage mechanisms) helped make the system easier to understand and specify.

Both CEPA and the tools that run under CEPA are important. Using one without the other does not yield benefits since one needs the control flow to determine which work to do, as well as the actual tools with which to do work.

The planning and scheduling tasks would have been difficult to implement. Because an ideal planning and scheduling tool would probably be embedded (like Oracle) since some tools as well as KI Shell would need to interact with it. Otherwise, information from the planning and scheduling tool would need to be constantly converted into a form that would be usable by KI Shell and other tools. This makes the API between KI Shell and the planning and scheduling tool more complicated. It also leads to the need for a new and complicated API every time a different planning and scheduling tool is selected.

A clear and precise Construction Plan for CEPA may have also helped direct the implementation team and would have given the specification and validation team a clearer set of assessment criteria since specific functions would have needed to be submitted at certain dates.

Uniquely describing stimuli and responses, by including the names of the devices between which the stimuli or responses move, made the categorization of stimuli and responses much more organized.

Using an appropriate file-naming convention can ease the task of storing state data, because related sets of files can be found by only knowing a partial name (suffix or prefix) relating a set of state data files.

Many box structure black-box subfunctions were similar, in that the conditions for responses were similar, although the responses (to the screen or to other devices as abstractions) were unique to each stimulus.

NOTE:

One disclaimer must be made about the specifications for the "Cleanroom Engineering Process Assistant (CEPA)." The CEPA specifications document < 37 on page 156 > is not a complete specification of the Cleanroom process. Cleanroom specifications have six volumes: (1) a *Mission Statement*, (2) a *User's Manual*, (3) a *Black-Box Specification*, (4) a *Black-Box Specification Verification Statement*, (5) a *Usage Profile*, and (6) a *Construction Plan*. The CEPA specifications contain partial versions of the first, second, and third volumes. Time constraints prevented these three volumes from being completed and prevented the other three volumes from being started. Therefore, the CEPA specifications should not be considered an example of a full Cleanroom specification. The three volumes should be viewed as prototype instances of those three volumes of a Cleanroom specification. Even from the portion of the Cleanroom process selected for defining the CEPA prototype, the benefits of Cleanroom specifications are apparent.

6.3 Validation of the "Cleanroom Engineering Process Assistant" Implementation

This section briefly describes the activity and records lessons learned during the validation of CEPA.

6.3.1 Brief Description

It must first be noted that the creation of the CEPA prototype entailed a four-step process: (1) determining objectives, (2) evaluating alternatives, (3) developing/verifying, (4) and planning for subsequent steps. The four steps were iterated a number of times. These iterations are consistent with the Spiral Model of software development conceived by Boehm < 5 >. The Spiral Model allowed the IBM team to increase its understanding of the problem at hand by following the same set of steps, while armed with more knowledge at each iteration.

The validation task is really twofold: validation of the preliminary versions of CEPA and validation of the final version of CEPA. The hope was that a concerted effort in noting and resolving problems in preliminary versions of CEPA would minimize the need for changes to the final version of CEPA.

The first task has been completed. A number of review meetings have been held between the specification and validation team and the implementation team, where the concepts of and specifications for CEPA have been discussed. Implementation options have also been discussed quite a bit to ensure that a consistent view of the implementation exists between the implementation team and the specification and validation team. In addition, these meetings afforded opportunities to assess the various versions of the CEPA prototype. CEPA was reviewed and validated against the specifications, with divergences being noted. User interface flaws and other shortcomings in CEPA were also noted. A major significant flaw in the CEPA implementation, which would have given CEPA a less than desirable "look and feel," was also resolved in one of the meetings. A Hypertext-like "look and feel" was preferred, given the capabilities of KI Shell. In this case the implementation team was able to take advantage of a feature of KI Shell in a way which had not been considered before. According to an implementation team staff member, this previously unrealized use of KI Shell was desirable and would change the way future implementations would be done using KI Shell.

The second task of validating the final CEPA was performed by employing coverage testing against the CEPA specification. The only true departure from the CEPA specification and the CEPA implementation was the implementation of state data management. The specification called for CEPA to automatically return all state data that had been checked out, back into the state data repository. It was not possible to develop this capability in the "S" increment. The problem of state data management can best be solved by the selection and integration of tools into the CEPA system to provide configuration and library management support.

6.3.2 Lessons Learned

Box structures makes it possible to perform verification at an early stage, since the assessment criteria were clearly defined.

It is better to find and fix problems as early as possible, since these corrections are less expensive than corrections made as a product nears completion. For that reason, a sequence of versions for the prototype was helpful.

User interface "look and feel" issues require solution domain evaluation and some experimentation with the prototype. It is difficult to predetermine what the system is supposed to look like before knowing the limitations of the implementation. Additionally, the precise user interface "look and feel" may be modified as validation of the final product occurs.

Due to the use of box structures, which was a clear description of the desired functionality, no major functional problems were found, only 'look and feel' related issues."

6.4 Major Lessons Learned from Case Study Preparation

This section contains major lessons learned by the specification and validation team during the IS-15 task.

Automating the Cleanroom process with CEPA focused quite a bit on the automation of the engineering tasks -- the basis for automation of the process.

The user interface is critical to make a software system usable, since the "look and feel" are features to which human users of CEPA react. Even after specifications are given, seeing what the software looks like is important, since that is a determining factor in making the software usable.

Specifications should be closely followed. It is often tempting not to follow them, since that avoids the need to understand another person's document. On the other hand, not following the specifications leads to development of software that may not be at all what is desired.

The need to have a CEPA that is as unintrusive as possible to the user is of utmost importance. A user-friendly CEPA will have been built when engineers using CEPA only see themselves following the Cleanroom process and do not realize they are also using CEPA. In other words, the process is visible to the users of the process, not the tools supporting the enactment of the process are invisible.

The importance of having an automated assistant to help in the adoption of a process was never so clearly recognized as it was when the specification and validation team participated in creation of the CEPA prototype. Individuals using traditional heuristic methods of software development will probably more readily move to a process with automated support than they would to a process without automated support. Having automated support for a new technology gives the new user a sense of security, in that there is something helping the user learn and take advantage of the process.

Having the box structure *Black-Box Specification* was very useful in determining when the implementation team diverged from the specification. Since the specification did not imply design, the specification and validation team and the implementation team did not waste time debating what design the specification was implying. Rather, time was spent working with the implementation team coming up with the best possible solution, given the time allocated for the case study.

The most important lesson learned was that it is both possible and extremely productive to use the formal design methodology inherent in Box Structure methodology when developing prototypes. The volume of documentation produced and the quality of it (in terms of correctness and consistency) would have been

difficult to produce by using any approach. Although such a design approach is formal, specifications produced using box structure black box notation were easily developed and quickly produced -- two traits that are often not associated with formal design approaches.

6.5 Use of Cleanroom Specification Techniques to Model Processes

For STARS Task IS-15, the Cleanroom process needed a suitable process modeling notation to allow the automation of portions of the Cleanroom process in CEPA. For IS-15, Cleanroom specifications, which are based on the concepts of Box Structures, were chosen to model the Cleanroom process. The use of Cleanroom specifications was beneficial and will be described in greater detail over the next few pages.

A specifications document for a process model must describe the process in sufficient detail and precision to allow the process engineer to implement the specification. A *Black-Box Specification* for IS-15 was created using *Box Structures*. In using a black-box description, all responses are described solely and completely as a function of stimulus histories. In this way all actions are clearly described without making implementation decisions, because all actions are described solely in terms of input and output. A sample black-box subfunction from CEPA specifications appears in Figure 15 on page 51. Additionally, since each stimulus has a different stimulus subfunction, the actions for every input are clearly described. Stimulus subfunctions are a good level of granularity for the process engineer, since the complete set of responses with full control logic for every input is given. The result is a description of what each part of the system does, accumulating into precisely what the entire system does. An additional perspective to describe the system was also needed, because users actually use the system by entering multiple stimuli and may think in terms of sets of inputs or transactions. Additionally, the relation between stimuli, such as where they are on and how they are accessed, is also important to see. In effect, not having the second perspective can lead to a classical case of "not seeing the forest for the trees." Cleanroom specifications account for the second perspective on the system with the *User's Manual*.

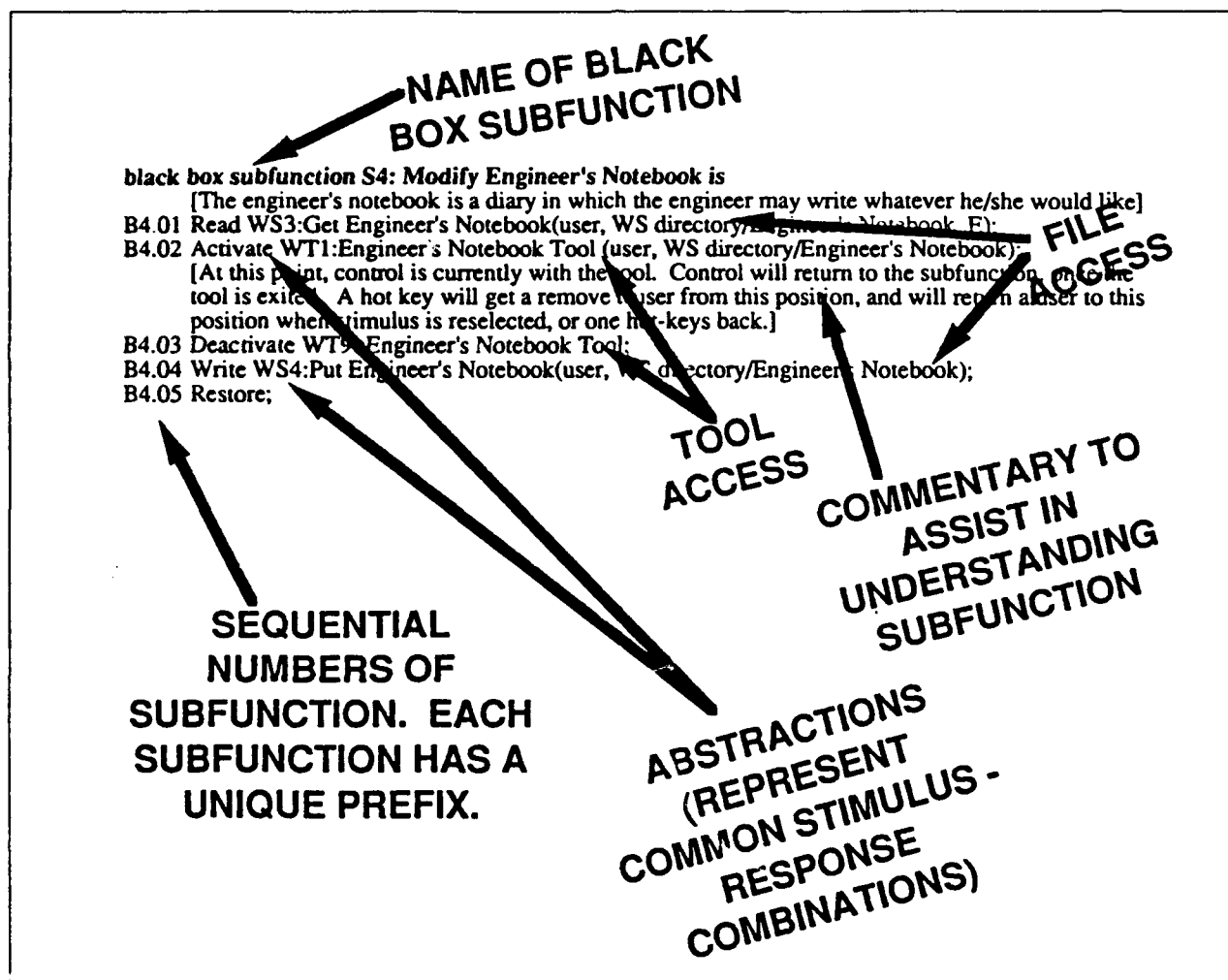


Figure 15. A Black Box Subfunction from the CEPA Specification.

The *User's Manual* describes the software system from the user's perspective. Each input corresponds to the selection of an option on the screen, such as a mouse click. The inputs available for each user are available in a textual or graphical form. In that manner a potential user can see, before implementation, what the 'look and feel' of the system will be like. The user interface is somewhat dependent on the implementation. For example, certain languages or features, such as X-Windows, allow the user interface to function in particular ways that may need to be considered when designing a product. Additionally, these features, such as OSF/Motif, require the use of additional stimuli, which are not fully described since they really do not affect CEPA. All inputs and their availability/location in the system are described in the *User's Manual*. The other feature of the *User's Manual* is a description of transactions, which are sequences of stimuli that lead to the completion of larger units of work. Transactions were shown graphically in the CEPA specifications to describe the sequence of stimuli that lead to the assignment, creation, and completion of a black box, for example. The transactions help users see how sessions or "day in the life" scenarios would work. These more closely describe the way they will use the system. Figure 16 on page 53 and Figure 17 on page 54 illustrate the inputs available for a developer and the transactions necessary to create a black box, respectively.

The *Black-Box Specification* and the *User's Manual* give two complementary views of the CEPA system. The *Black-Box Specification* gives a precise description of the system in terms of stimuli histories and responses. The specification completely describes what the system does. This level of detail is necessary for the process engineer who will be implementing the process. The *User's Manual* complements the *Black-Box Specification* by describing the system from the user's perspective. This document describes the system in

terms of how it looks and how it will be used. By reading both the *Black-Box Specification* and the *User's Manual*, a user clearly understands the functionality and the "look and feel" of the software system to be implemented.

Box structures were also used to describe the Cleanroom process for STARS task IR-70/E, creation of "*The Cleanroom Engineering Software Development Process (SDP)*" <36>. In that document the Cleanroom process was divided into a set of 25 processes. In preparing the Cleanroom Engineering SDP, black, state and clear boxes were used. The state and clear boxes served to describe what information was stored, and how it was stored. The clear box also helped to clearly describe the process, in terms of engineering tasks and conditions. Engineering tasks are the actual steps that engineers, or teams of engineers, follow to do work. Conditions serve to show the control flow (in terms of entry and exit conditions) between different engineering tasks and between processes and between engineering tasks and processes. The conditions included completion conditions, which are detailed checklists of actions that must be completed before the process itself is considered complete. Figure 18 on page 55 and Figure 19 on page 56 display a sample Cleanroom process from *The Cleanroom Engineering Software Development Process*."

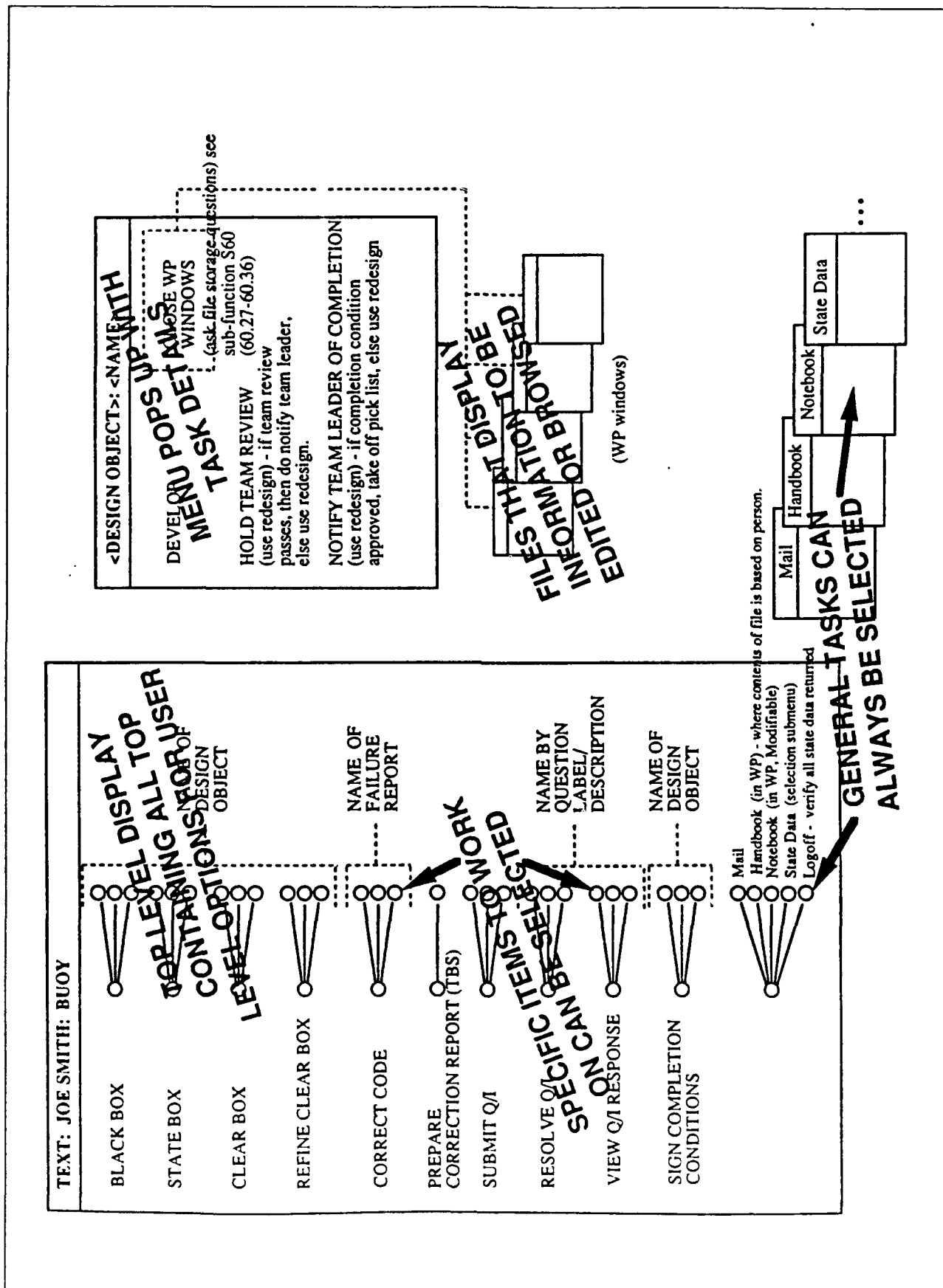


Figure 16. Developer Screen Format.

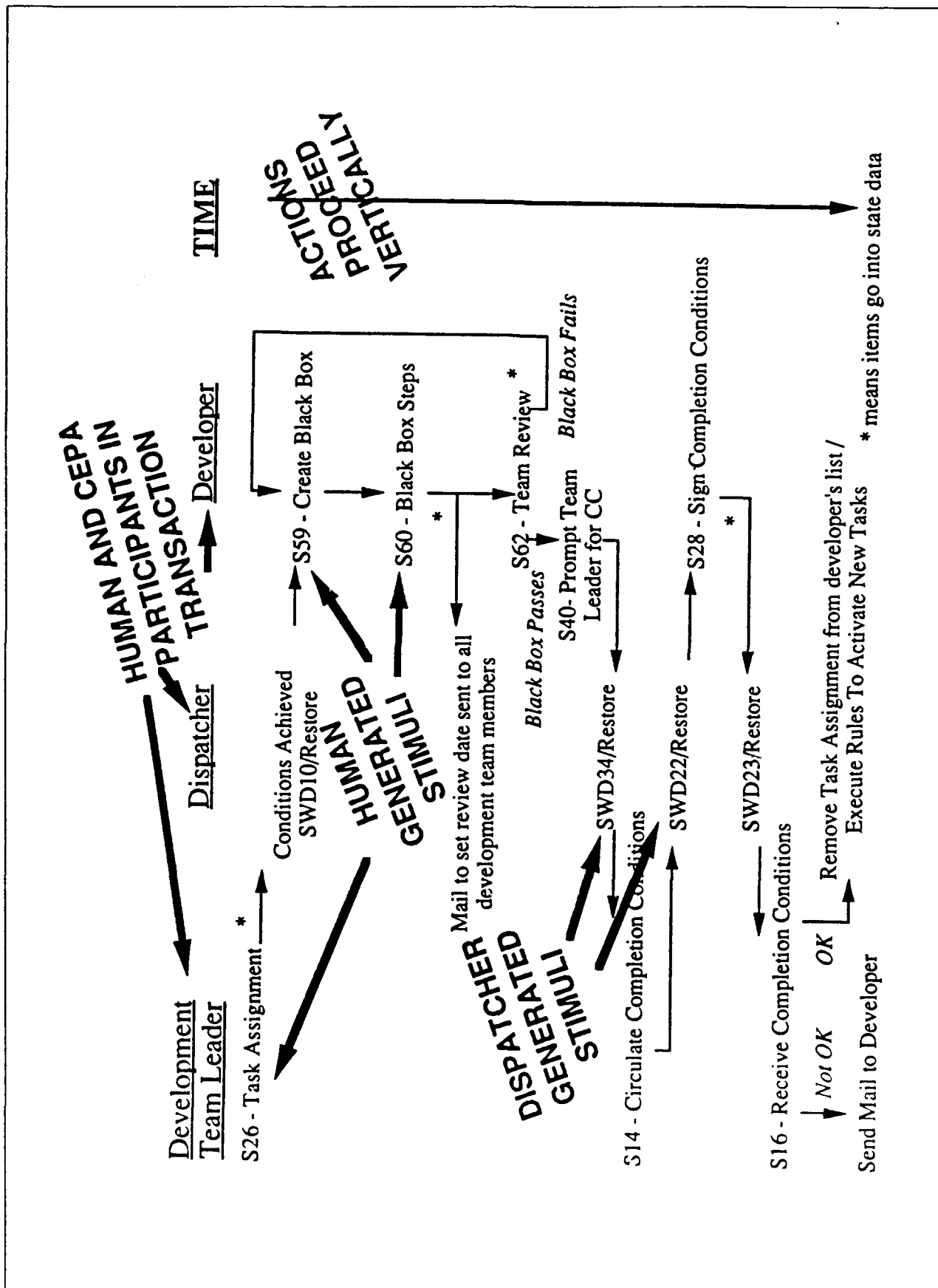


Figure 17. Using CEPA Facilities to Perform a Black-Box Task.

Process E19 - Develop Certification Plan and Tests For Increments 1..i

Process Summary The Develop Certification Tests For Increments 1..i process completes the preparation for the certification of the software product. A usage-based testing approach is developed, which will allow for usage testing to be conducted on the software. The process is illustrated in the figure below, and is also described in greater detail below.

Outer State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes

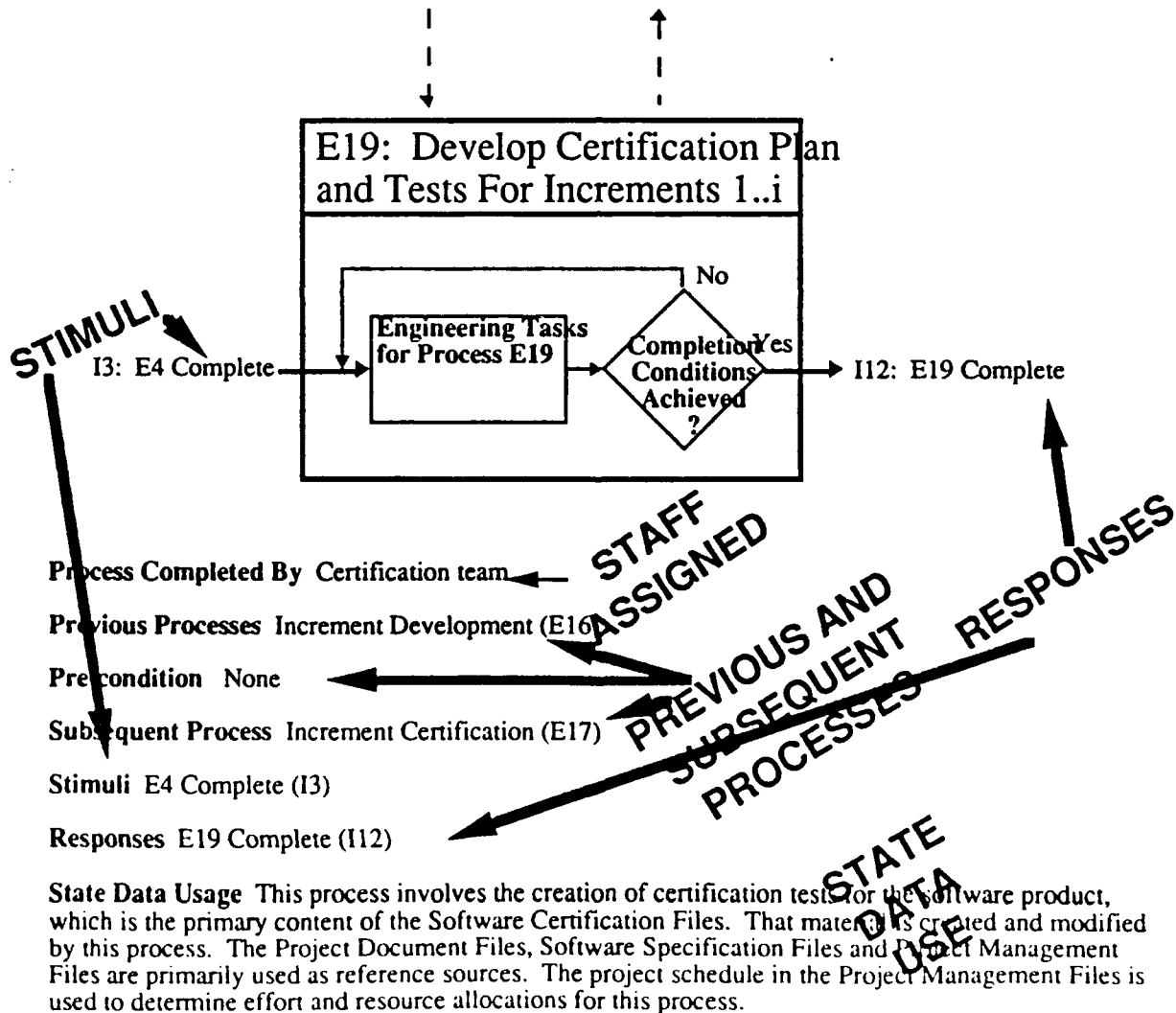


Figure 18. Sample Process from Cleanroom Engineering Software Development Process (1 of 2).

Process Description The Software Specification Files contain the Usage Profile Volume, which presents, in the form of a matrix, the transition probabilities which define the probability of a user moving from any one program state to another program state. This matrix was developed by applying a Markov model to all the identified program states that the software can reach. Additionally, the Usage Profile Volume contains the stimuli to the system and the corresponding distributions for each stimulus in each state. Using the matrix and the stimuli information, the Certification team develops test cases by completing the following sequence of engineering tasks until the completion conditions are achieved:

1. Modify the usage profile according to the Construction Plan.
2. Determine the requirements coverage of each state and each state transition.
3. Determine how many test scenarios are required to test the increment to the desired level of reliability, given the expected rate of failures.
4. Specify the sampling scheme to be used to guide the testing for the increment. This includes the decision of whether both control flow and data will be randomized or just control flow.
5. Develop test scripts, or test script generators that list the stimuli and stimuli values which complete the program state transitions.
6. Develop test scenarios by creating random state transitions. Use the test scripts or script generators to generate each program state transition. This task may include the random generation of data.
7. Based on the test scenarios generated in task 6, compute expected outputs for each test case. The expected results form the basis of comparison for validation of test executions.
8. Set reliability targets and failure limits for the increment.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process:

1. Do test scenarios reflect the operational profile of the software to be tested?
2. Can the results of executing all test scenarios be validated?
3. Have test passage criteria for the increment (such as number of test scenarios, failures or reliability goals) been determined and corresponding test information generated?
4. Have expected results been generated, and passage criteria determined for each test case?
5. Have sufficient test scenarios been generated to certify the increment to the desired level of reliability, given the expected rate of failure?
6. Is the increment complete according to the items listed in the Construction Plan (Volume 6 of the Specifications)?
7. Has all state data in the Software Certification Files been correctly added, changed or deleted?
8. Have all pertinent reviews for this process been completed?
9. Have all action items generated during reviews that pertain to this process been completed?
10. Have all information to be preserved been placed in the correct state data?

Keyword References

- Test Script, Test Script Generator, Test Scenario
- Section 10
- "Engineering Software Under Statistical Quality Control," IEEE Software, November 1990 (Cobb, Mills)
- "Statistical Quality Control of Software System Development," SET Course

PROCESS
DESCRIPTION

MEASUREMENTS

COMPLETION
CONDITIONS

REFERENCES

Figure 19. Sample Process from Cleanroom Engineering Software Development Process (2 of 2).

The previous discussion has given the general benefits of using Cleanroom specifications. The specific question to be discussed was: "what are the benefits of using Cleanroom specifications to model processes?" For the CEPA prototype, the IR-70/E report, which described the Cleanroom process, was used as a basis for the *Black-Box Specification* and *User's Manual* for CEPA. These two documents define the part of the Cleanroom process that will be implemented/controlled by the CEPA software for the Cleanroom Software Process Case Study. For this reason, the chain of argument will justify first the use of box structures for the Cleanroom process, then the use of Cleanroom specifications to describe the software for process modeling.

The box structures description in IR-70/E presents all of the information necessary to describe a process in a convenient manner. Relationships between the processes were clearly described. Inputs to and outputs from each process are listed, in terms of stimuli and responses. Entry and exit conditions for a process are listed. In this case the entry conditions were completion of previous processes or preconditions' being true or false. Exit conditions are the lists of Completion Conditions for each process, which also serve to validate that engineering tasks were all properly completed. Project data (state data) that is used by a process is also clearly listed. This serves to relate project activities to the creation, acquisition, or use of information during the project. The list and organization of engineering tasks is also given. The engineering tasks are the actual units of work for engineers to complete with the Cleanroom process. Additional information that is kept in the clear boxes for the Cleanroom process includes a process summary, a list of people who are to complete the task, measurement data, and references to keywords.

In analyzing the processes found in the IR-70/E document, one sees that the necessary information to describe the process is available. Most of this comes as a result of creating the black- and state-box views of each process before making the final clear-box description. In that manner, the function of each process was understood, as well as the information created or stored during that process. Some minor details are also added in the clear box, to complete the description of the process. The hierarchy of the boxes was also determined using the Box Structures algorithm, which created lower level processes as they were needed, rather than creating them randomly and trying to find a heuristic way of connecting them.

Using two volumes of the Cleanroom specifications to specify CEPA was eased by the fact that the description of the Cleanroom process had been done using box structures. The specification could proceed from a fairly rigorous problem description. The details of the clear boxes also served as the basis for the black-box subfunctions because where the engineering tasks left off in IR-70/E is where the specifications for CEPA began. In effect, the *Black-Box Specification* volume drove a portion of the IR-70/E document to a deeper level.

A portion of the Cleanroom process was selected for the IS-15 task case study. The software parts of the process were then specified in the CEPA specifications. The specifications were written quickly and were easy to modify, which is impressive given the amount of detail found in the document. This was a result of the use of box structures for both the IR-70/E document and the CEPA specifications. The two perspectives on the system could be used in combination to understand the functionality of CEPA as well as the "look and feel" of the system, without assuming and accepting numerous implementation decisions. That is an extremely valuable benefit.

As a result of the STARS IR-70/E and IS-15 efforts, the benefits of using box structures and Cleanroom specifications to model processes as well as to specify software have been realized. In terms of completeness and clarity, box structures are extremely useful. Additionally, the savings in terms of cost and time of using this form of specifications are also significant.

One measure of the effectiveness of a specifications document is the resulting implementation. UES, using the specifications, was able to create a fairly large and detailed the CEPA software system within the constraints of the IS-15 schedule. This was attributable to having a Cleanroom specification that clearly described the system to be implemented and the power of K1 Shell to quickly implement the specification. The result is a the CEPA system, consistent with the specification, that was developed within the IS-15 schedule.

Intentionally left blank.

7.0 Software Process Enactment Experiment and Demonstration Preparation

This section provides an overview of the "Cleanroom Engineering Process Assistant" (CEPA) concept developed as part of the "Cleanroom Software Process Case Study" and describes the implementation of the CEPA prototype system, which permitted us to examine the process of developing a system to support the enactment of a defined process. Finally, this section will provide instructions for installing the CEPA system and a scenario for demonstrating the CEPA prototype system.

7.1 CEPA Demonstration System Description

One of the goals of the STARS program is to instantiate a Software Engineering Environment (SEE) with program development and support tools, to facilitate DoD software development. CEPA is an integral component of a SEE. To establish the relationship between CEPA and a SEE, we will review the basic elements of a software project and role of a SEE in performing a software project.

7.1.1 Software Engineering Environments

Software development is organized around projects. A project is authorized to develop an item of software to accomplish some mission that has been assigned to the software. A software project is an organized undertaking to develop an item of software and all its associated deliverables. Software projects are completed by people (software engineers) who have been assigned to the project. The staff are provided with resources to assist them in completing the project. These resources include:

- The software development processes that are to be used to complete the project. The assigned processes define the techniques, practices, tools, methods, and data required to perform the process.
- Training materials and other reference materials for the assigned processes, techniques, practices, and tools.
- Workstations (including the associated system software and associated interworkstation communication facilities) on which the project engineers individually and collectively perform their work on the project.
- Software to operate on the workstations that helps the project engineers utilize the software development processes that have been assigned to the project.

The resources included in the above list are integral parts of a process-managed Software Engineering Environment (SEE) and form the basic requirements for an environment to support software engineering.

Each project must have an instance of each of these resources. The instances may vary only modestly from project to project or they may vary a good deal. The beginning point for establishing a SEE for a project is the definition of the process and its associated tools and procedures. Once the project process is defined, the other parts of the project SEE can be assembled. Therefore, there must be two parts in the software that supports a SEE. The first part is the portion of the software that the process engineer uses to define the process and then to assemble the project-specific SEE. The second part or the SEE software is the software used by project participants (managers and engineers) as they perform the project. The demonstration system only provides capabilities of the second type of activity, but the work in specifying the software has pointed to a way to develop software to support the first type of activity.

7.1.2 CEPA and Software Engineering Environments

The SEE concept is that when a project is initiated, a process engineer(s) defines the process that the project is to utilize. The defined process and its use of associated practices, tools, and project data are recorded in the software operating on the workstations. In this way the project engineers and managers can easily use the defined process and as a result concentrate on intellectual tasks of designing the software.

The view point taken by CEPA is that the processes to be utilized by a project are defined by the Cleanroom principles < 1, 3, 7, 11, 12, 18, 19, 20 >. The specific process to be used by a project can be documented by process engineers using black-box functions to define the responses that engineers and/or teams of engineers should produce at each state of the process. When using black-box functions, the function responses are defined in terms of stimuli that the process has already defined. In the CEPA view the results of previous project design decisions reside in the project state data.

Before defining CEPA, the Cleanroom process was defined at the highest level in the IR-70 Extended project < 37 >. Then to develop CEPA, the Cleanroom process was defined in greater precision by including complete details for how to design and certify software using the Cleanroom process. This definition was documented in terms of black boxes in the CEPA specification < 36 >.

To put the CEPA work into perspective, the following repeats the SEE resources as defined in the previous section and points out how the CEPA work relates to each of the resources:

1. **The software development processes that are to be used to complete the project. The assigned processes define the techniques, practices, tools, methods, and the data required to perform the process.**

In developing the CEPA specification, black-box functions were used to document the detailed process to be followed. In developing the specifications, the viewpoint taken was that of a process engineer documenting the process for others to follow. Since this project was only to define and develop a prototype system, only a portion of the Cleanroom process is defined in the CEPA prototype specification. CEPA experience indicates that black-box functions are ideally suited for defining and documenting processes.

This experience points in a promising direction for developing a production version of CEPA for the project initialization portion of a SEE. A process engineer will define the project process in terms of black-box functions that the software can read and understand. Presumably this will not be a difficult task, since generally a process engineer will be fine tuning previously defined processes. As a result, there will be high levels of reuse of process definitions. The software then uses these process specifications to tailor itself to perform the process as defined. It seems feasible to develop software that performs in this way.

2. **Training materials and other reference materials for the assigned processes, techniques, practices, and tools.**

The CEPA prototype provides on-line access to the *Cleanroom Engineering Software Development Process (SDP)* handbook. Since the software engineers will be using box functions to define specifications and software, they will be able to easily understand the process definition as written in terms of black boxes. This precise, easy to understand process definition facilitates the transfer of process knowledge.

3. **Workstations (including the associated system software and associated interworkstation communication facilities) on which the project engineers individually and collectively perform their work on the project.**

The CEPA specification assumes the existence of workstations connected over an easy to use network. Another assumption is that the workstations have screens of sufficient size to reasonably hold images of many related documents. Additionally, it is assumed that the workstation system software is capable of executing multiple applications and that it facilitates people using multiple applications such as "cut and

paste" and "live copy and paste". The workstations are connected to a project state data server, where all state data is maintained in a controlled environment.

4. **Software to operate on the workstations that help the project engineers utilize the software development processes that have been assigned to the project.**

The software needs to provide three main functions. First, the software must coordinate all interpersonal work and individual work to guide the engineer through the process providing automatic access to all the correct tools and data. CEPA is intended to be this coordinating software. Second, the software must provide the tools that the engineers need to use in designing the software, which is the object of the project. The CEPA specification defines some 30 tools that software engineers need to access. Third, the software must perform all project management planning and control activities.

The prototype CEPA provides (1) access to the some 30 prototype tools that developers and certifiers require, (2) facilities for engineers to communicate with each other and work together in teams, (3) facilities for engineers and managers to communicate about task assignments and project status, (4) an accurate status report of the project by recording the status of completion conditions, so that only tasks for which all prerequisite tasks are fully complete are dispatched, and (5) access by managers and team leaders to project management and scheduling tools.

7.1.3 CEPA: An Overview

The mission of the *Cleanroom Engineering Process Assistant* is to enable software development organizations using the Cleanroom process to produce high-quality products while increasing productivity.

This mission is accomplished by providing on-line assistance to all members of the software engineering team utilizing the Cleanroom process. The Cleanroom Process (CP) has been shown to facilitate the development of essentially defect-free programs and to increase the development team's productivity. CEPA will have the following missions in aiding members of the development team to use CP:

1. Minimize realization productivity losses, that is, reduce the time lost because supporting activities are not properly coordinated. CEPA will significantly improve the probability that all of the prerequisites, tools, and data that an engineer needs to do a task are available with no wasted time on his or her part.
2. Make it easy for the engineer to follow the Cleanroom process and thereby obtain all of its benefits.
3. Enforce the Cleanroom process in the most unobtrusive way possible by being user-friendly.
4. Make it easy for all levels of management to plan, schedule, and control all project tasks and to ensure that the required reviews and verifications take place.
5. Make it easy to collect all required metrics for statistical control of the development process and better estimates of development time and cost.
6. Update on-line state data, the data needed to develop the product, and make them immediately available to all members of the development group.
7. Provide direct, on-line access to standards, tutorials, and other aids.
8. Improve formal and informal communication between the members of the group.

The net result of an organization's equipping its engineers with CEPA is having in place a repeatable, defined, managed, and optimized software development process according to the SEI Maturity Capability Ratings <13>. By using the Cleanroom Process, supported by CEPA, organizations can expect to develop essentially failure-free software with much less cost than is currently required to produce failure-rich software.

⁹ "Live copy and paste" refers to the ability to cut tables, text or figures from different sources, and have them automatically updated, when the source files are modified.

7.1.4 Using The CEPA System

Upon logging into CEPA, one is presented with a screen corresponding to his or her role on the project. A role corresponds to the responsibilities an individual can have on a project. For example, one can be a development team leader or a certification team member. The next few pages, will describe "a day in the life" of a development team leader. The screen that a development team leader is presented with, upon logging in, is shown in Figure 20 on page 63.

Figure 20. Development Team Leader Screen Format.

What first must be understood is that a development team leader is also a developer. This is logical, because there is no need for a full-time "manager" for the development team, and thus, the team leader will also participate in the actual development work.

As a team leader, the development team leader can assign tasks. This is done in two ways. By selecting the "Assign Tasks" option on the bottom of the screen, new tasks, such as creating the black, state, and clear boxes for a module (a module is the black, state, and clear box along with all clear box-refinements for a code component) can be assigned. Tasks can be preassigned in that they are assigned to be completed, but are not available to be selected by a developer until specific preconditions are achieved (such as the completion of a previous box). It is also possible that the team leader will only assign some tasks, for example, only assigning the black box. The other assignments (or a change in a task assignment) for the module can be selected by again selecting "Assign Tasks" but this time selecting a specific task, all of which are visible to the right of the "Assign Tasks" option. If a specific task has not been assigned and is available to be assigned, it will appear at the top of the screen, at the "Tasks Waiting" option. A task assignment selected there can also be modified.

Team leaders also manage their team. In the CEPA prototype the only two management capabilities available for the team leader are viewing project metrics and updating the schedule. Other capabilities can be added as CEPA moves towards a production system.

The developer tasks, such as creating or modifying black, state, or clear boxes, refining clear boxes or correcting code, are available to be selected only when a specific design object is assigned or a failure report is to be corrected. When a specific design object or failure report is selected, a menu appears that lists the steps that are necessary to complete the activity. A selection of a development step (the first option typically available) will open up a number of windows that have supporting information and windows that will contain the files to be developed. Files are created and organized by the CEPA. Upon exit, the supporting files are discarded (since they are supporting information, the instances do not need to be saved), while the user is given the option whether to save the edited files in the state data repository or elsewhere. Whenever files are edited, the user is given this sort of option. The benefit is that the state data is kept up to date automatically. This removes most of the administrative responsibility from the user. One can leave this set of menus by selecting an option from the main development team leader's screen, which is also visible.

Options on the menu for a developer task in addition to development include holding a team review and notifying the team leader that completion conditions need to be distributed. The team review now displays only the edited files (in a nonedit mode) and another window which contains a file for the review minutes, that will be stored in state data. Production versions of CEPA will use some sort of groupware that will allow multiple individuals to participate interactively on the workstation. Once the team review passes and the developer is convinced that this task is complete, the option on the menu is selected, and the team leader is sent a notification.

The team leader sees the notifications next to the "Circulate Completion Conditions" option. Selecting the specific object for which the notification was sent distributes a completion condition form to each member of the team. These forms can be read, marked, and signed by selecting the "Sign Completion Conditions" option for the specific design object. Once completion conditions are signed by all team members, the team leader sees the design object's name next to the "Receive Completion Conditions" option. Selection of this option will allow the completion condition sheets from each team member for the design object to be viewed, which will allow the team leader to determine whether the task has been completed. Upon completion of the review of the forms, a pop-up screen appears that gives the team leader the option of considering the object complete or incomplete. If the complete option is selected, the design object's name will be removed from the task it was next to and may lead to new options appearing elsewhere on this or another user's screen. If the incomplete option is selected, then the option remains where it was and must continue to be worked on. When the pop-up screen disappears, the development team leader's screen is visible again.

The next few sentences will illustrate what occurs on the screen. If a black-box "buoy" was being created, "buoy" would appear next to the black-box option on the screen. Selecting "buoy" would allow editing, and

when the task was completed and signed, "buoy" would disappear from next to black box. Of course, the task to create the state box for "buoy" would be dependent on the completion of the black box, so "buoy" may now appear next to the state-box option on the screen.

Questions and/or issues are a way of circulating and archiving the process of increasing the knowledge or understanding in topics related to the project. Questions can be submitted, resolved (when a question/issue is submitted for a user to resolve), or viewed (when the response is completed and the question is returned to the submitter). As the question or resolution is submitted, it is stored in state data as a part of the permanent archiving of the project.

The development team leader is also involved with Certification Reports, which are the notifications from the Certification team of the status of the certification process for an increment, and Correction Reports, which are notifications of corrections made by the development team to code being certified.

Finally, each engineer is given a number of general activities that can be done. Sending and receiving mail is one of these options. Another is viewing the Engineering Handbook, which contains explanations of engineering or management tasks for the roles assigned to the user. For example, the development team leader would have developer and development team leader entries visible in the Engineering Handbook. The Engineer's Notebook is a diary that contains any information that a user may want to preserve. Files in the state data repository can also be viewed by selecting the "View State Data" option. Finally, a user can log off from CEPA. The logoff option presents a user with a list of all information taken from state data and gives the user the option of returning each file or keeping it signed out for a future session. All five of these options are made available continuously. For example, the Engineering Handbook can be visible while the development team leader is assigning tasks. This would make the team leader's job easier, by not forcing him or her to memorize all of the policies in assigning tasks.

The CEPA system and the tools running under CEPA handle many administrative and communications tasks. CEPA allows an engineer to focus on the intellectual tasks of creating and verifying black, state, and clear boxes (specification and design data).

7.1.5 CEPA Features

The *Cleanroom Engineering Process Assistant* (CEPA) will automate and facilitate the features of the Cleanroom process that pertain to organizing, planning, controlling, measuring, and directing a software engineering project.

The basic CEPA features include the following:

- The automation of some of the planning, scheduling, and task assignment activities (activities that are considered a part of process E7 - Maintain Project Schedule -- and its precondition, C2 - Schedule to P. Modified/Published) <37>;
- Authorizing (enacting) tasks as they are selected to be worked on;
- Facilitating and archiving important communications between team members;
- Meticulously maintaining state data. (Using Cleanroom, software and other deliverables are developed by continual modifications to state data, not by creating a number of intermediate deliverables. Since the state data is continually used and modified, maintenance is critical.)
- Controlling access to and modification of project state data;
- Facilitating the assessment of and sign-off of the completion conditions; (The commencement of a subsequent process is based on the fulfillment of all completion conditions for the preceding processes.)
- Controlling the interrelationships among tasks;

- Controlling and facilitating the access to state data and tools in relation to tasks. (A staff member executes a task when assigned.)

Through CEPA, the staff member is assisted in accessing state data and a tool, if needed, before starting work on a process. There is little direct communication between processes, where a response from one process is a stimulus to another; the output of a task is put into state data and acquired by succeeding tasks as needed. All information necessary for a task to be completed is found in state data.

Roles Represented in CEPA

The roles that are a part of CEPA are the following:

- Program Manager
- Software Engineering Manager
- Specification Team Leader
- Development Team Leader
- Certification Team Leader
- Review Team Leader
- Specification Team Member
- Development Team Member
- Certification Team Member
- Review Team Member
- Process Engineer
- Librarian
- Business Manager
- CEPA Administrator

The CEPA prototype will have the following roles' activities:

- Software Engineering Manager
- Development Team Leader/Development Team Member (one engineer who has both roles)
- Certification Team Leader/Certification Team Member (one engineer who has both roles)
- Development Team Member
- Specification Team Leader/Specification Team Member (since Specifications are assumed complete, one of the other participants can assume these roles also).

(The other Cleanroom roles will show on role selection menus, but a selection of such an item will present only a text description of the role.)

Processes Available to Roles

In terms of menu options that the roles can select, the following are available for all of the roles mentioned above for the prototype case study demonstration scenario:

- Do assigned tasks (for each role, the task types will be described in greater detail below)
- Submit Question Issue

- Resolve Question/Issue
- View Answer to Question/Issue
- Circulate Completion Condition List
- Mail
- Modify Engineer's Notebook
- View Engineering Handbook
- View state data
- Logoff.

For each role, the screens are similar to the development team leader's screen that was previously discussed. The descriptions of the tasks are also similar. For that reason, only the screens for the other roles will be presented on the subsequent pages, as (Figure 21 through Figure 27). Interested individuals may want to refer to the CEPA Specifications < 36 > for a more detailed description of the capabilities for any role.

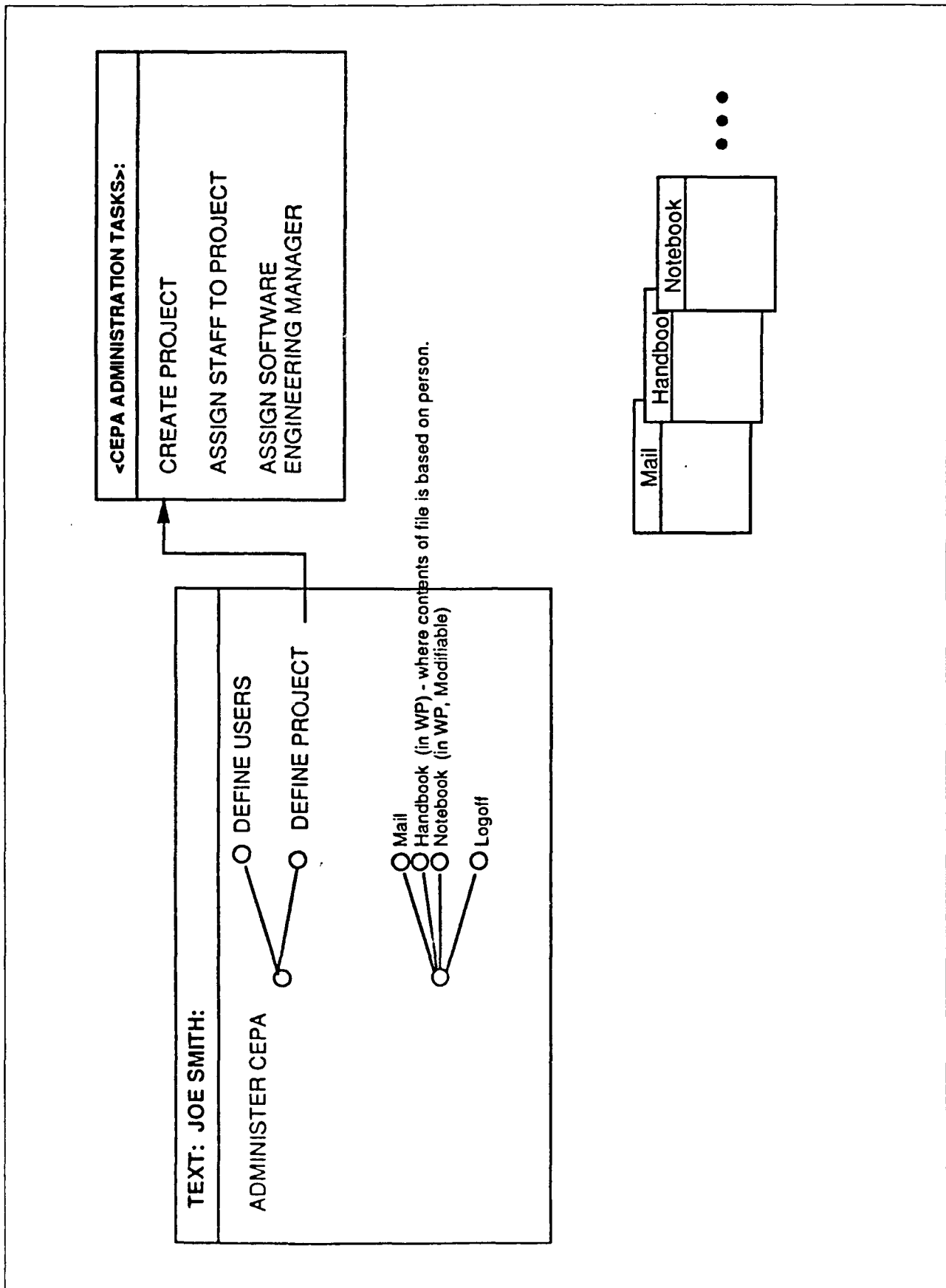


Figure 21. CEPA Administrator Screen Format.

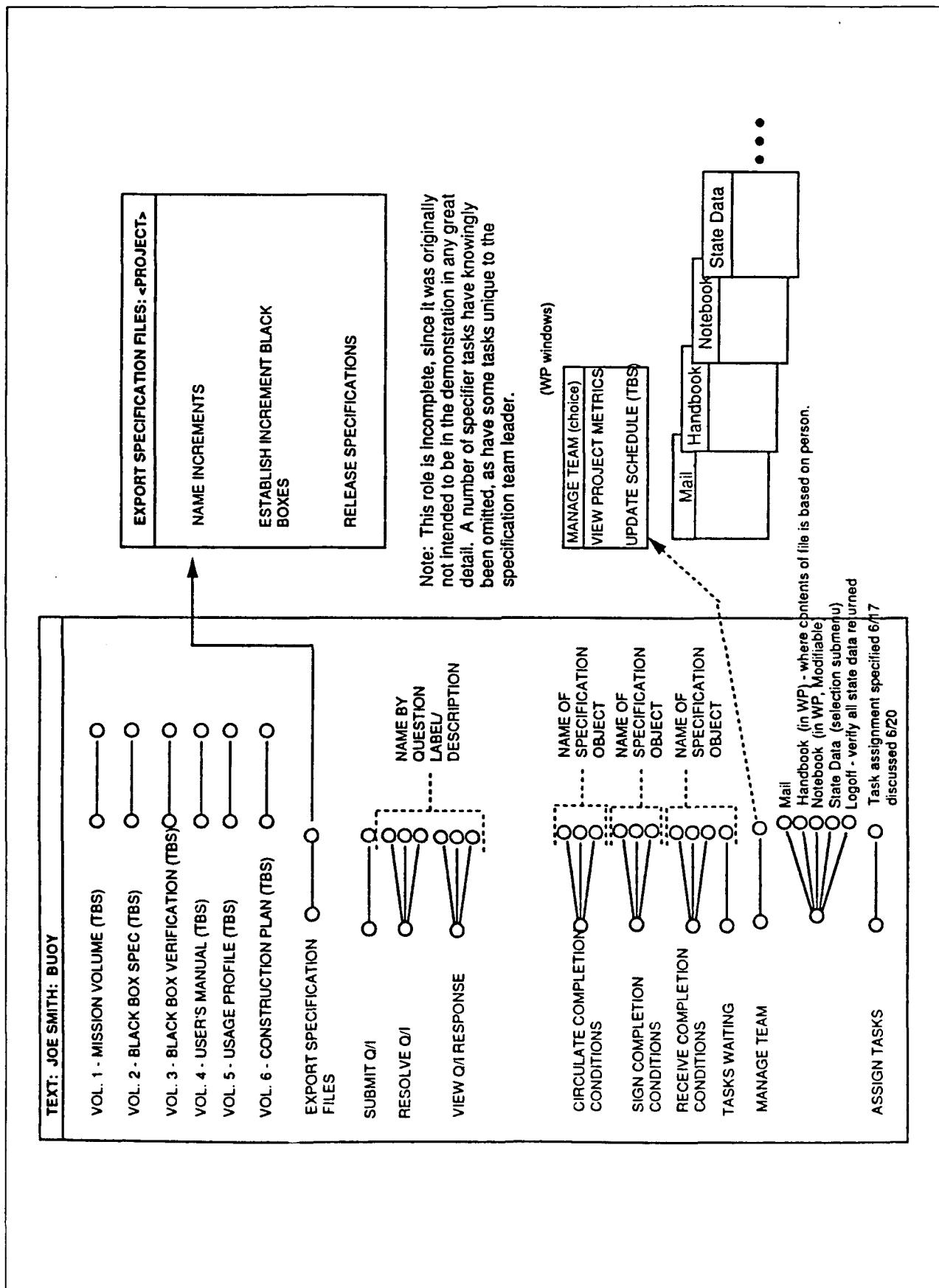
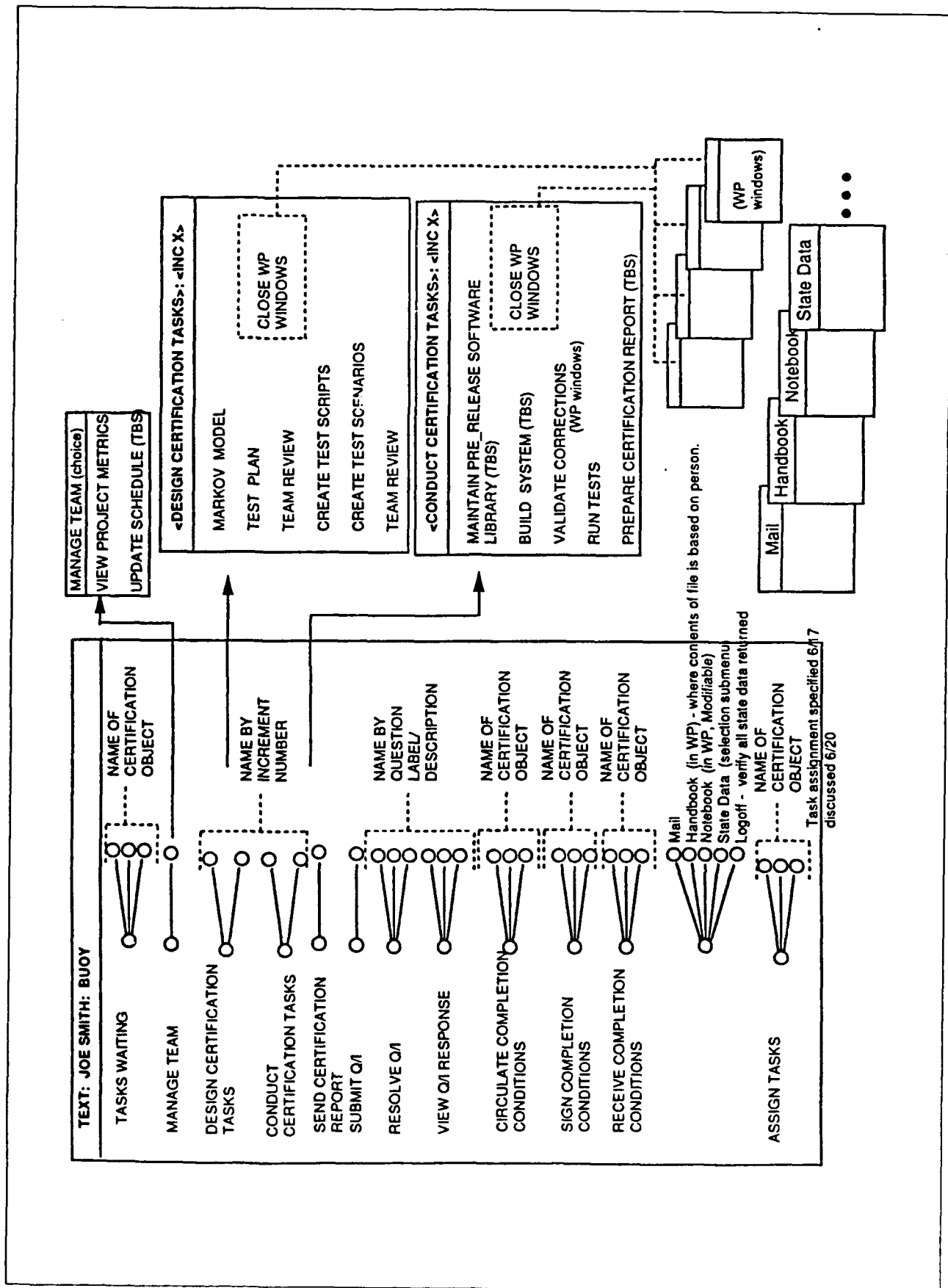
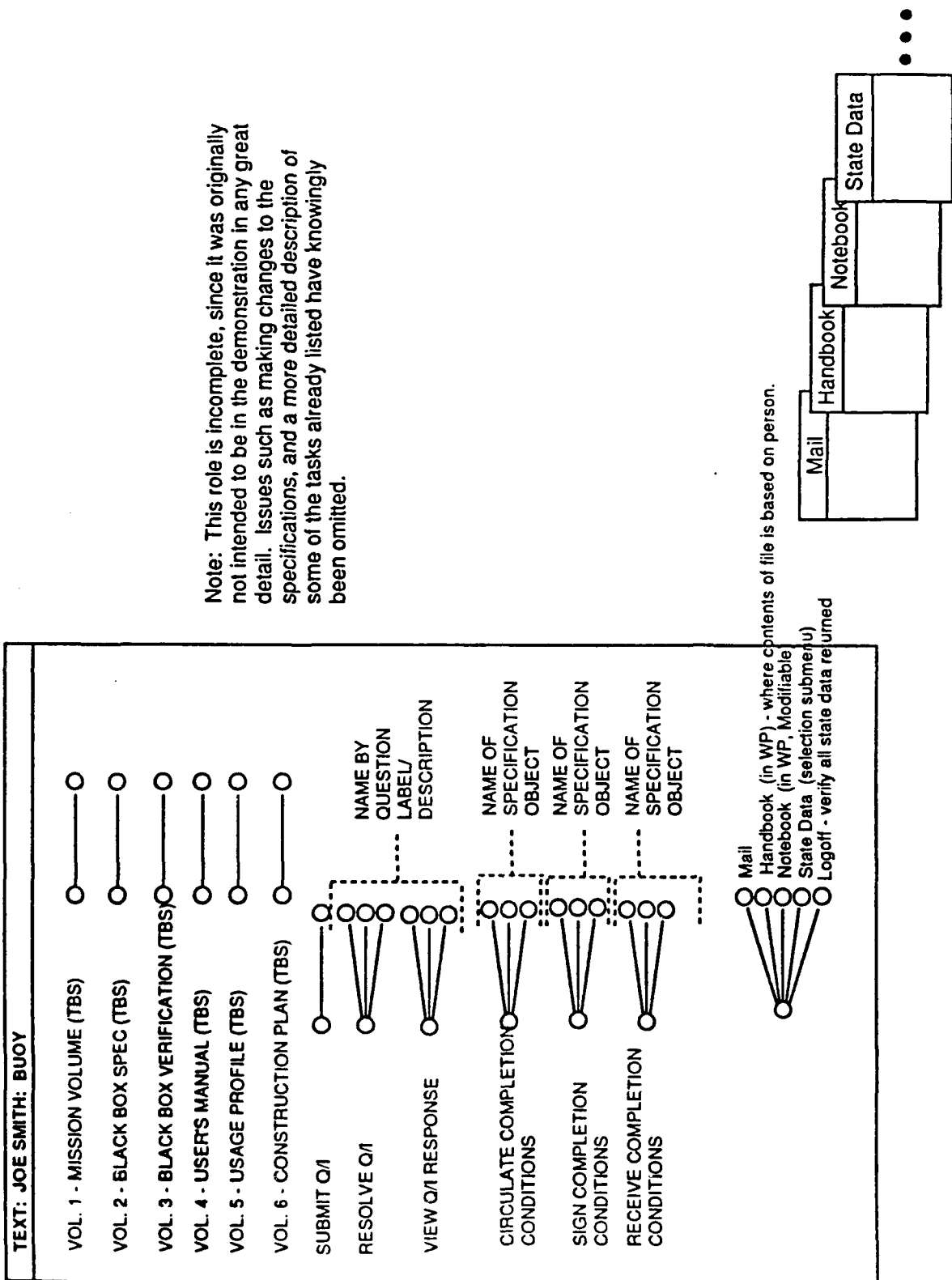


Figure 23. Specification Team Leader Screen Format.





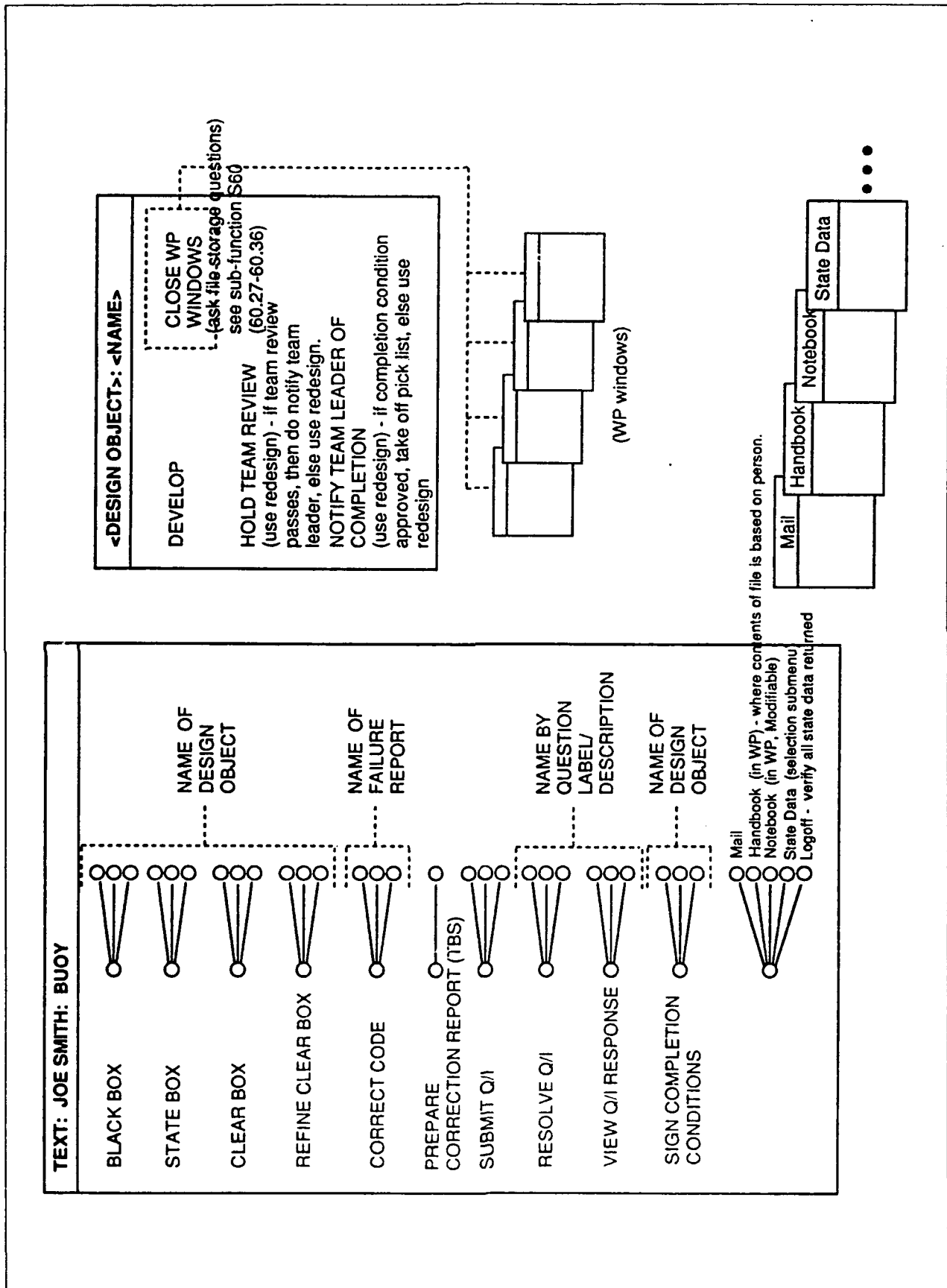


Figure 26. Developer Screen Format.

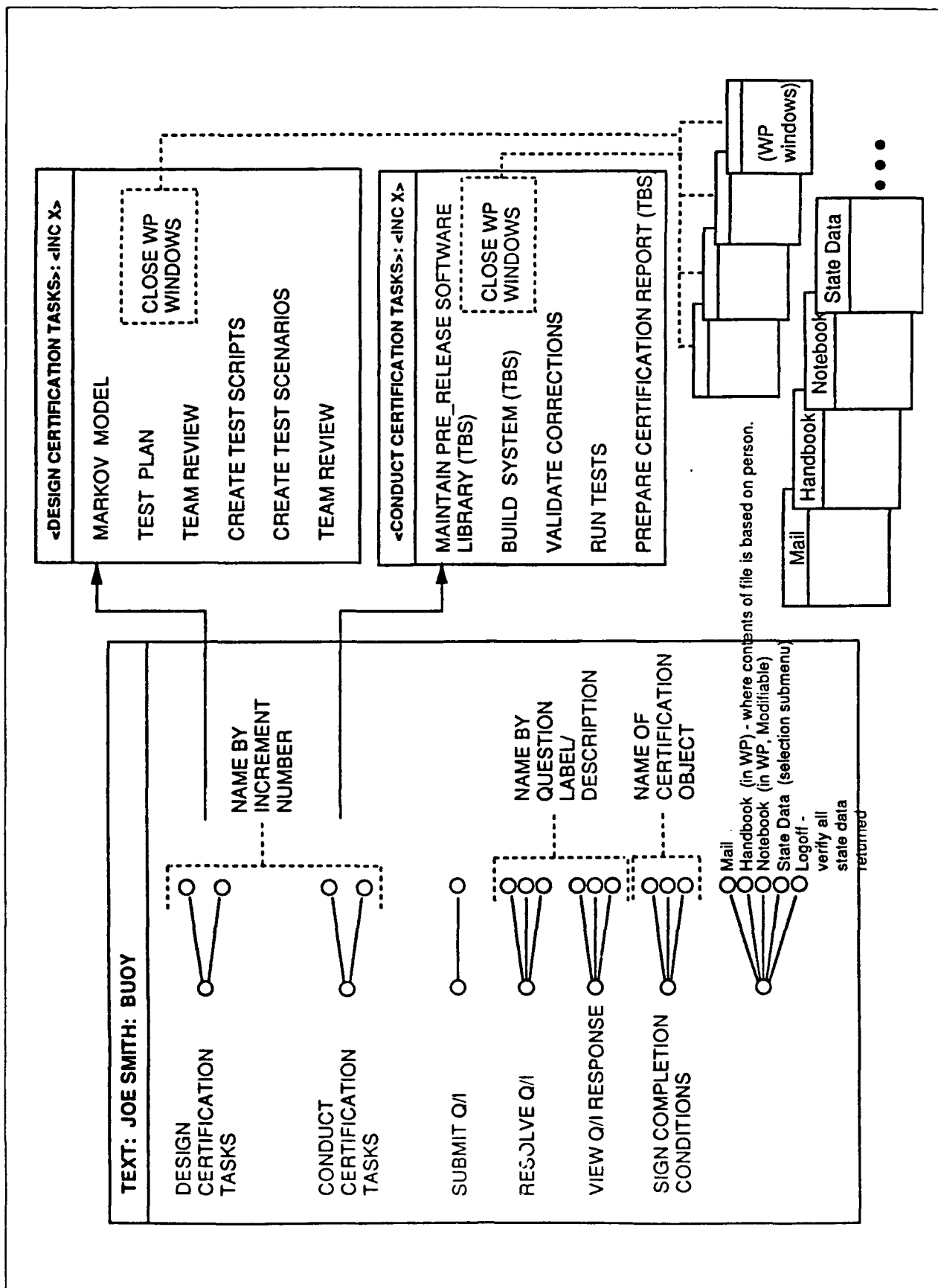


Figure 27. Certifier Screen Format.

7.1.6 CEPA Tools

The CEPA system handles a number of the control flow issues pertaining to the Cleanroom process. Additionally, tools must be made available to assist engineers in completing their assigned work. These tools are not actually a part of CEPA, since different tools that satisfy a function can be interchanged. The key point of CEPA is to provide a useful suite of tools to the engineer to support his or her Cleanroom engineering activities, as they are needed in following the Cleanroom SDP.

Tools may be invoked to assist an engineer in completing a task. In the prototype CEPA, many of the necessary tools are mimicked by WordPerfect. For a production CEPA, actual executable tools are required. These tools assist an engineering in completing a specific part of the Cleanroom process. The tools do not ease the creative tasks for engineers, but they ease organizational and administrative issues (actions that can be automated), so that engineers can focus completely on the creative tasks. The types of tools necessary for a CEPA include the following:

- Box Structure editor to facilitate the design of black, state, and clear boxes, as well as refinements to clear boxes.
- Verification aid to focus the analysis and reporting that is involved in the verification process.
- Markov analysis tool to assist in creating the usage profiles.
- Test scenario generator to automate the process of selecting and generating test cases.
- Correctness aide to assist in process assessment.
- Statistical tools to aid in computing failure and mean-time-to-failure data.
- Project Planning and Scheduling tools to aid in developing plans and schedules.
- Process Management and Control tools to aid in the day-to-day flow of the project entities through the defined processes.

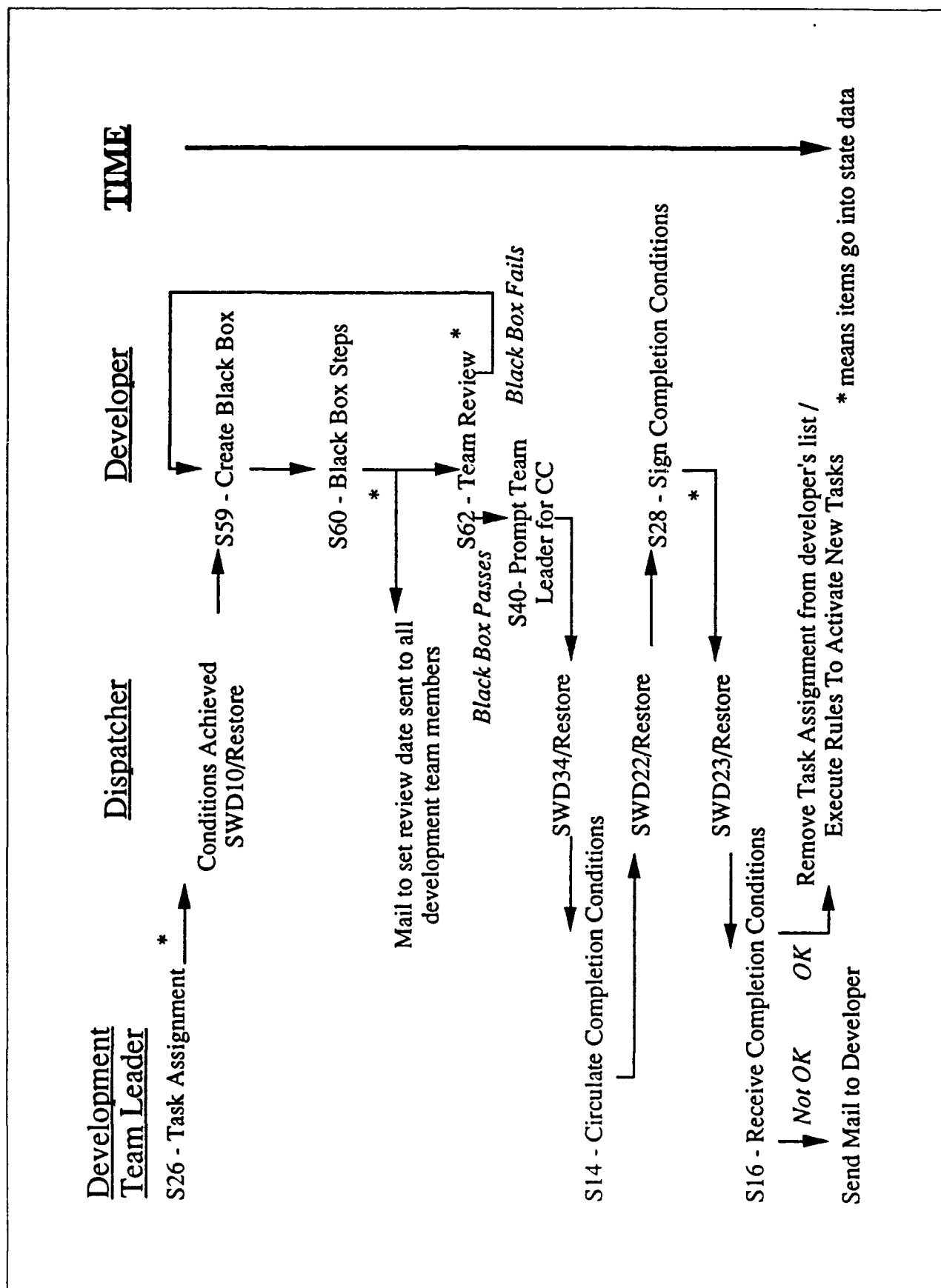
Each tool would obviously be tailored somewhat to fit into the Cleanroom process. The intent would be to build CEPA in as general a manner as possible to eliminate the need to tailor a tool for CEPA. For example, issues such as reading from and writing to state data would need to be handled by CEPA, not by the tool.

7.1.7 Using CEPA (continued)

CEPA is used by selecting options visible on a screen that is defined for a specific role. The actual options have been listed in the sections above. Conceptually, selecting an option can be considered to be selecting a work unit or task to work on. Selection of any option will cause an external tool or feature (the name given to functions to complete tasks that are a part of CEPA, not an external tool) to be activated, with the necessary corresponding information. Completion of a task will allow necessary data to be saved and will return a user to the general screen for that user's role.

It must be emphasized that although any specific unit of work can be started by selecting a single screen option, to create and complete a unit of work multiple screen options may need to be selected by a number of users. For example, creating a black box really entails the sequence of creating a task assignment to create a black box (done by the development team leader), selecting the black box to create, editing the black box (both done by developer), and conducting a team review for the black box (done by team). Once these steps are successfully completed, the developer must prompt the development team leader to distribute the completion conditions for the task, then upon distribution by the team leader, the entire development team must sign completion conditions, which are returned to the development team leader to decide whether the task has actually been completed. At that point, the task disappears from the list of tasks to be completed. Of course, the completion of one task may be the precondition for another. For these reasons, tasks should be viewed as a sequence of activities when considering the life cycle of a specific task, from creation to com-

pletion. The sequences of activities necessary to complete different types of tasks, with specific references to the CEPA specifications (such as stimulus numbers), appear on the subsequent 9 pages (Figure 28 through Figure 36).



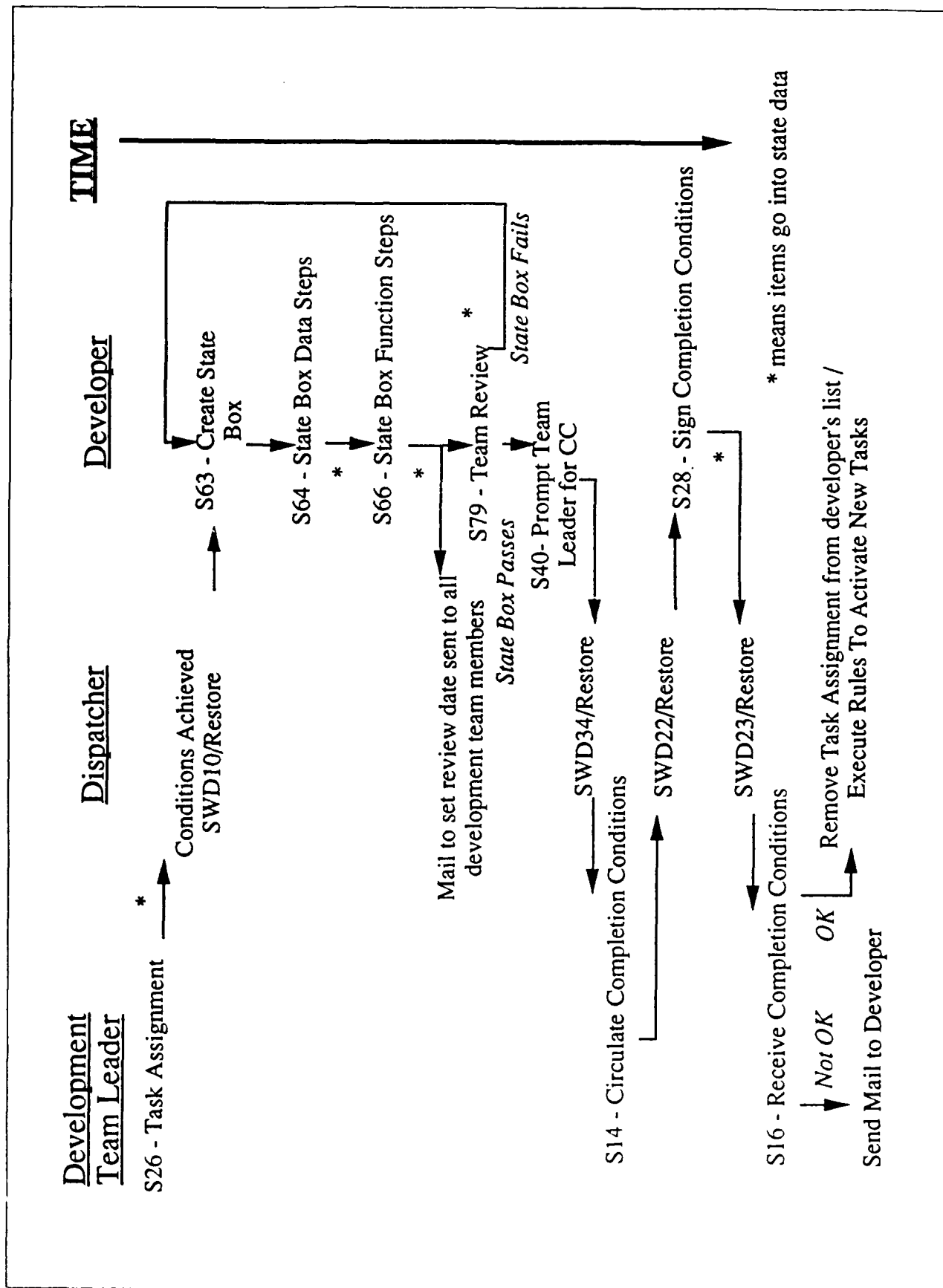


Figure 29. Using CEPA Facilities to Perform a "State Box" Task.

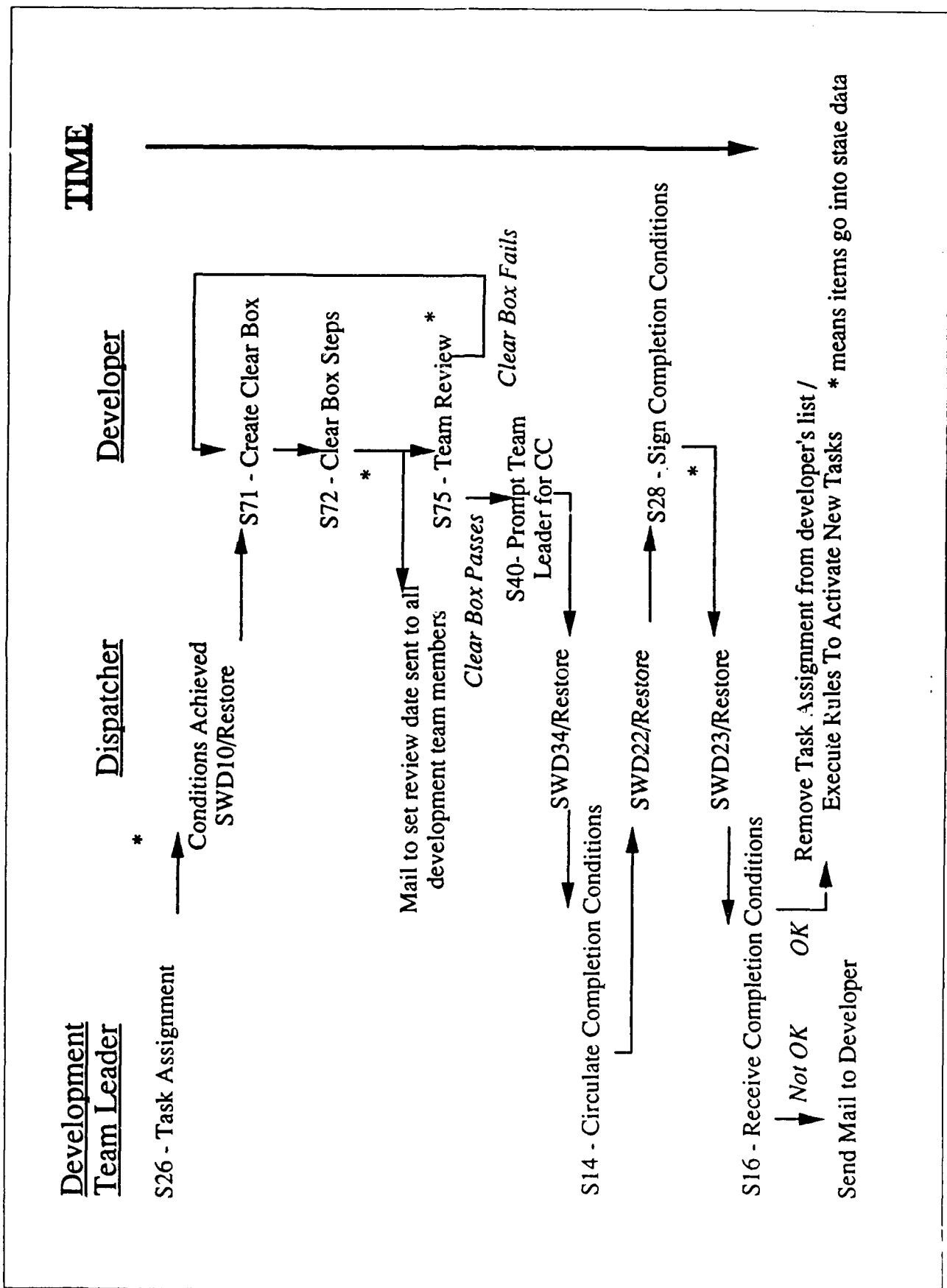


Figure 30. Using CEPA Facilities to Perform a "Clear Box" Task.

AD-A255 945

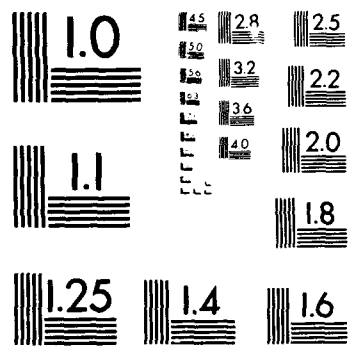
SOFTWARE TECHNOLOGY FOR ADAPTABLE RELIABLE SYSTEMS
(STARS) PROGRAM SOFTWARE (U) IBM FEDERAL SECTOR DIV
GAITHERSBURG MD W H ETT 30 SEP 91 03705-001 XC-AFSC
F19628-88-D-0032

273

UNCLASSIFIED

NL

NPMS



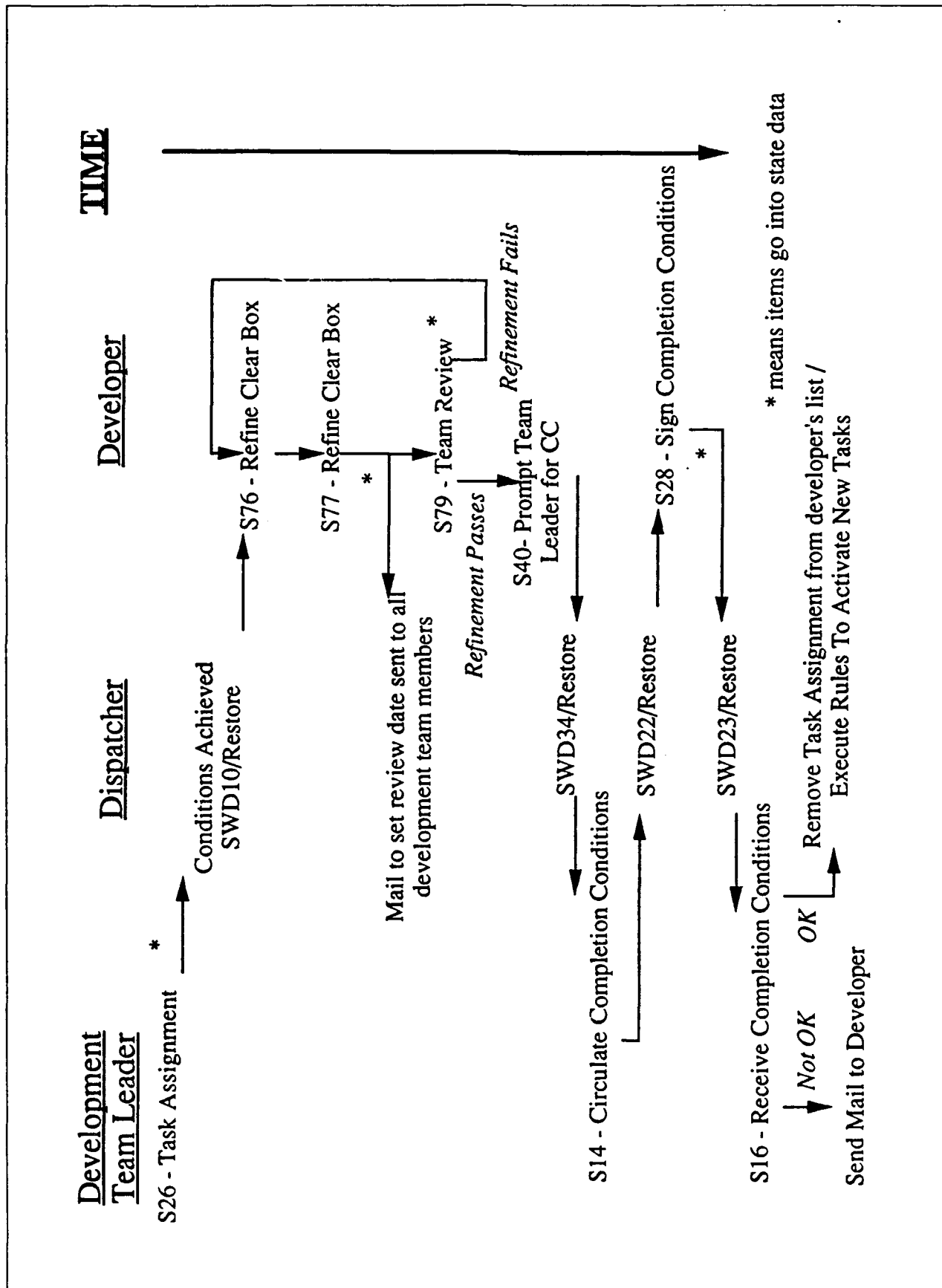


Figure 31. Using CEPA Facilities to Perform a "Refinement" Task.

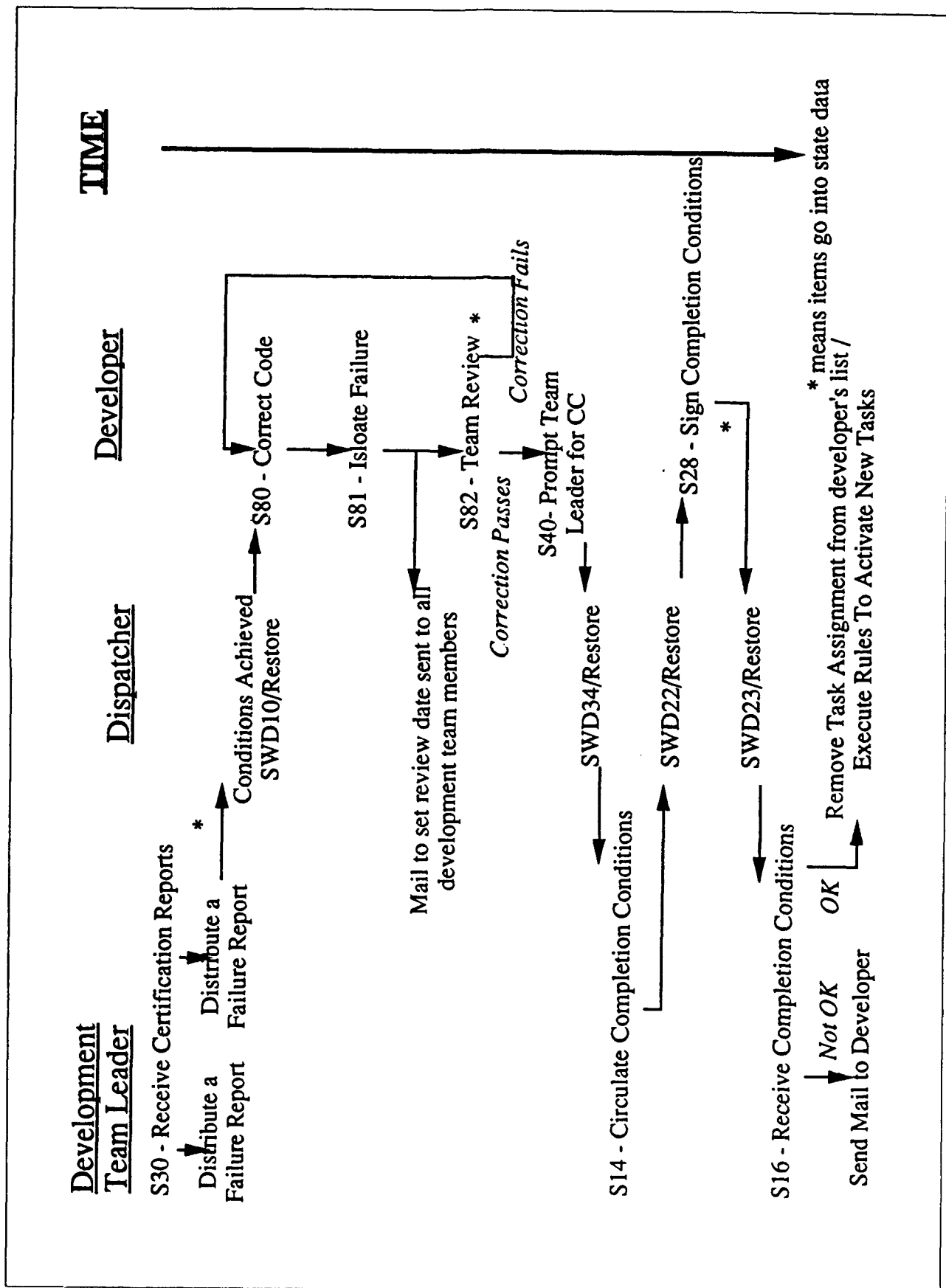
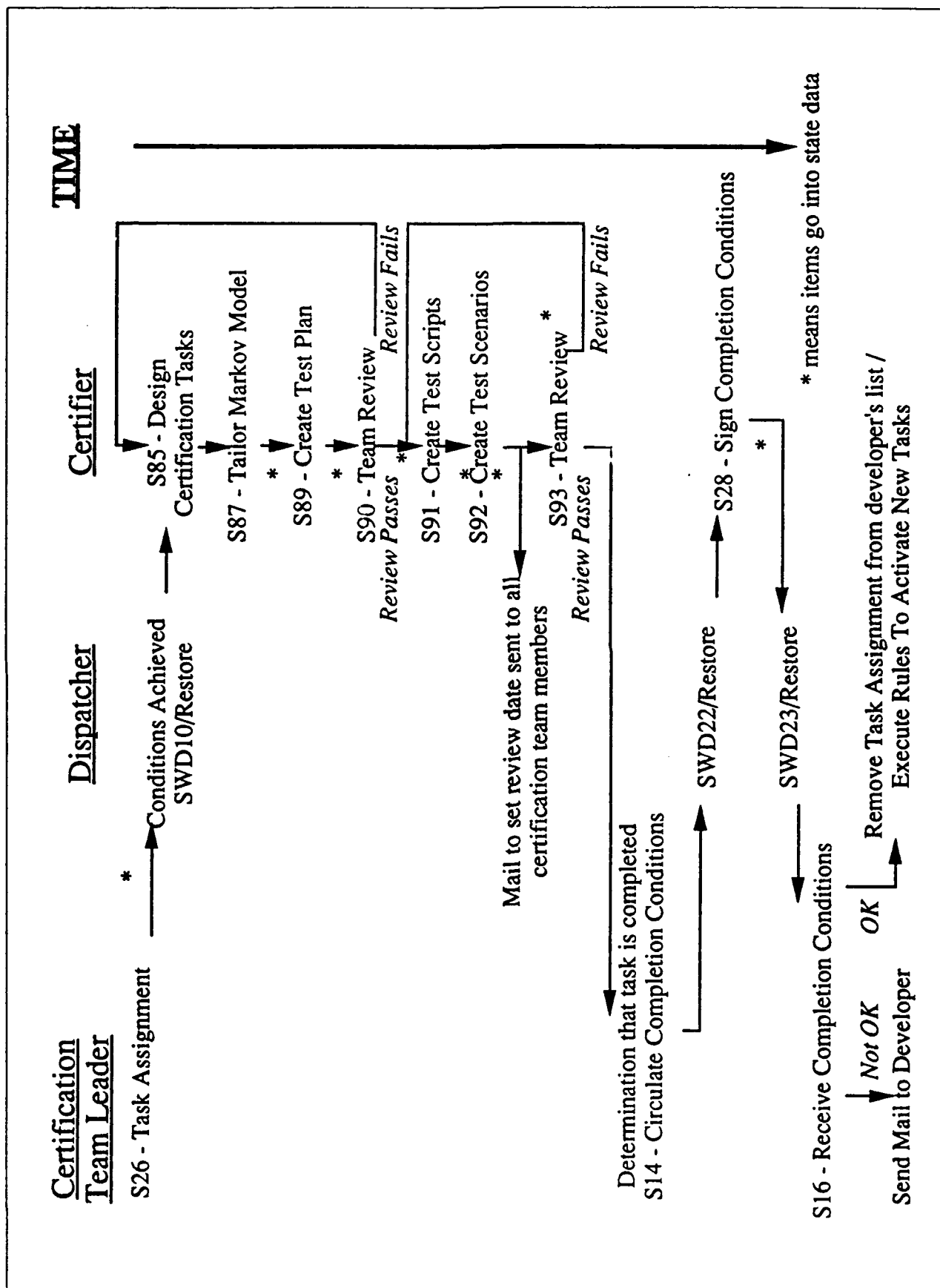


Figure 32. Using CEPA Facilities to Perform a "Correcting Code" Task.



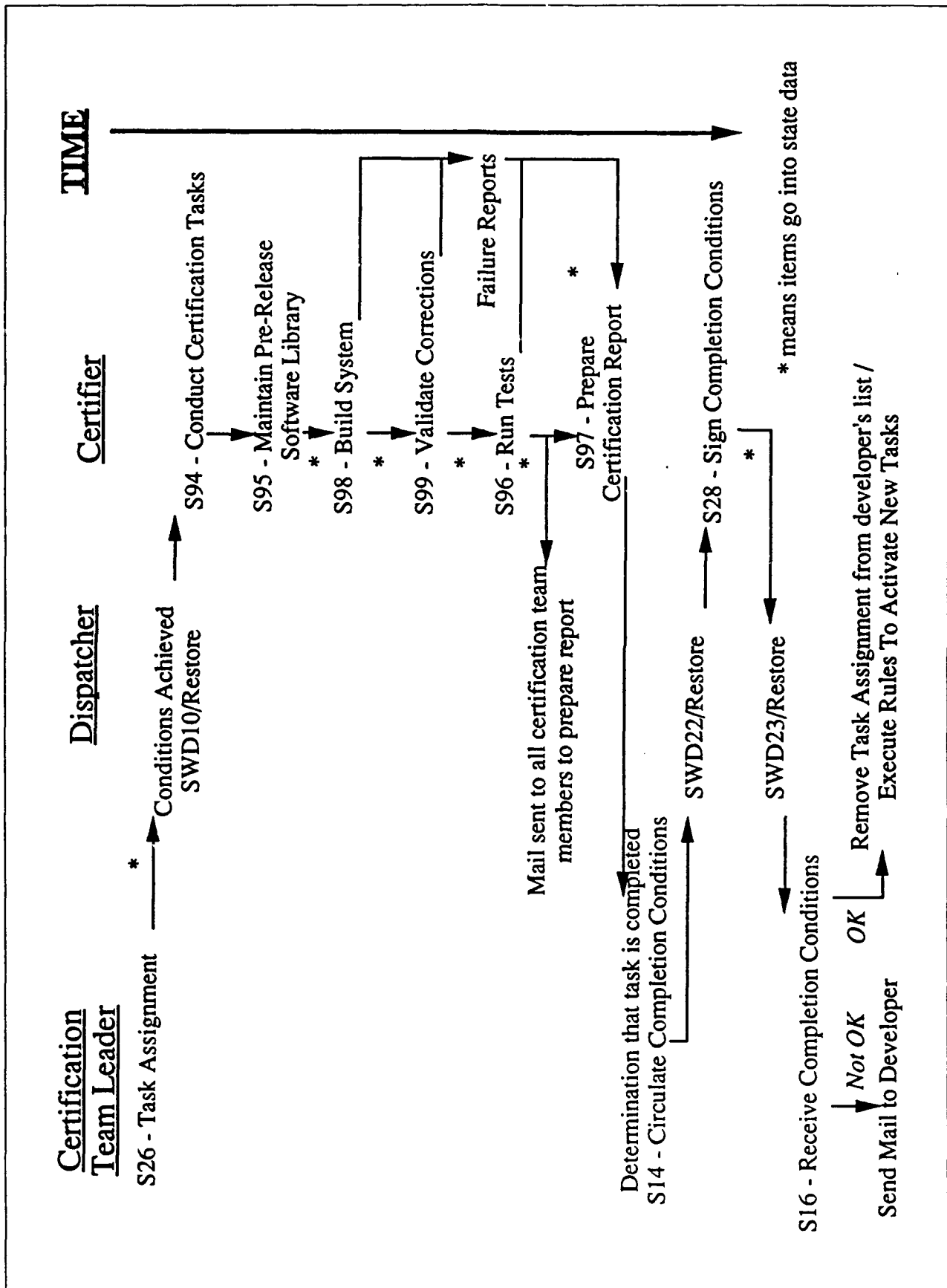


Figure 34. Using CEPA Facilities to Perform a "Conduct Certification" Task.

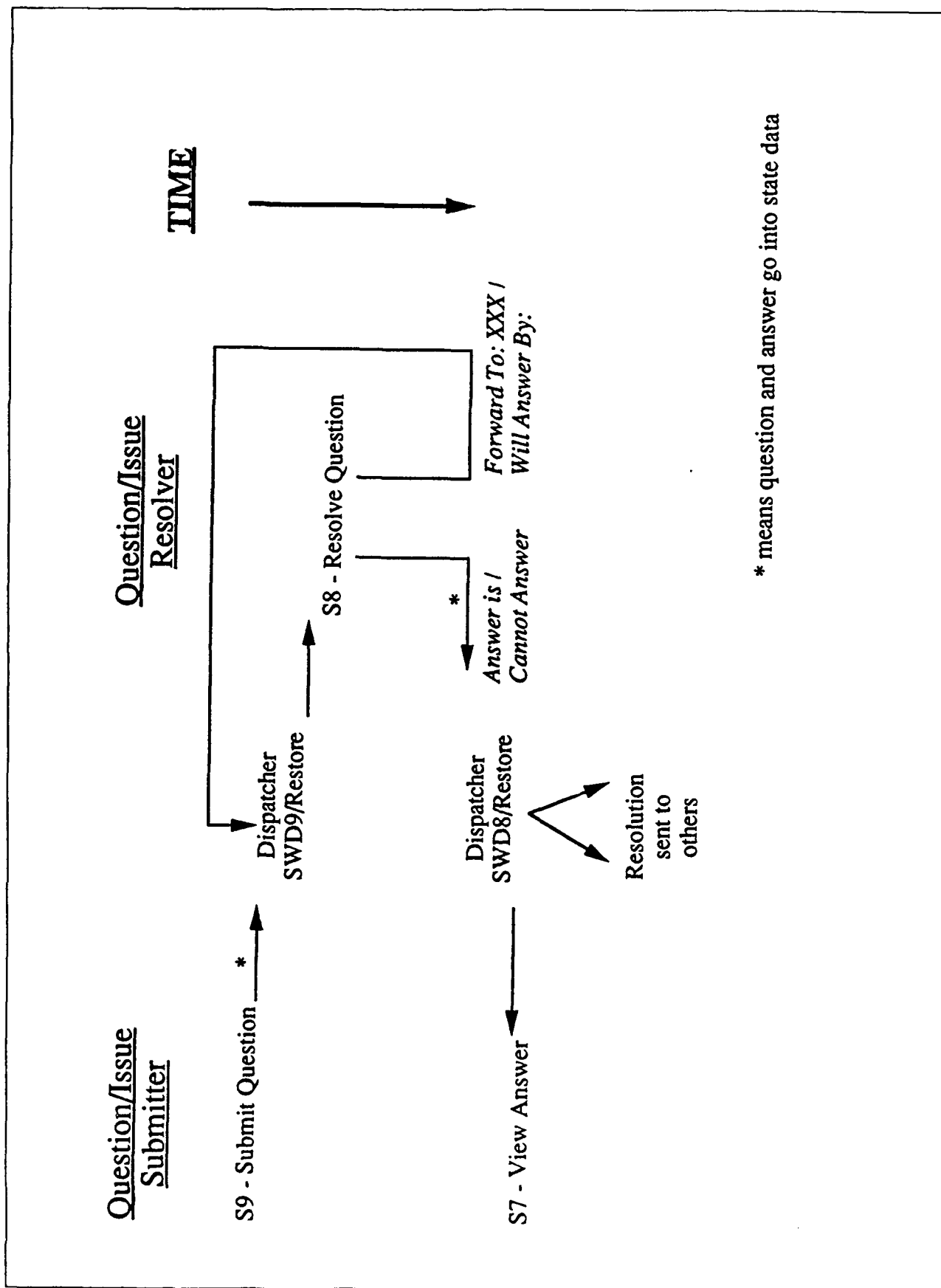


Figure 35. Using CEPA Facilities to Perform a "Submit, Resolve a Question" Task.

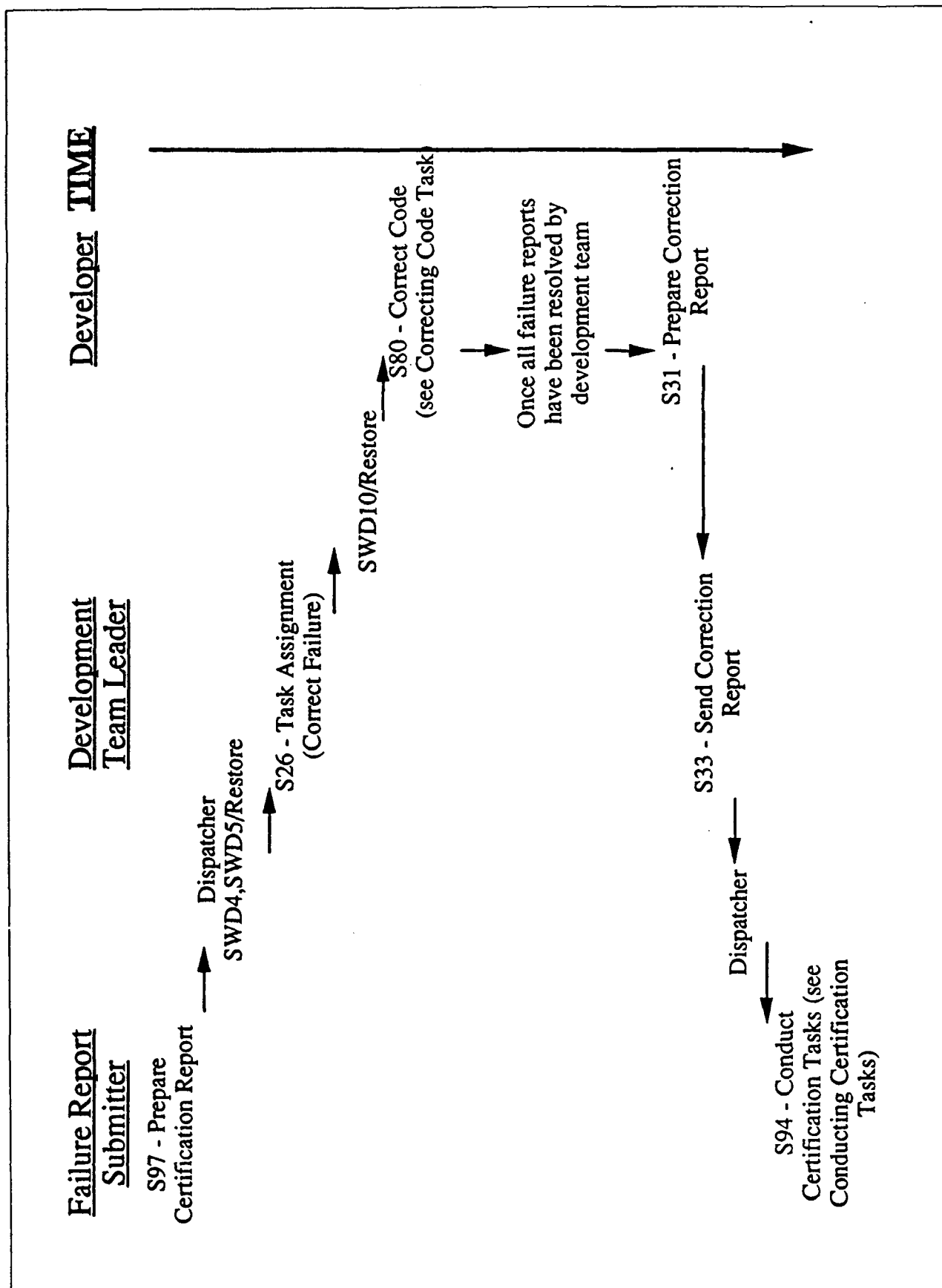


Figure 36. Using CEPA Facilities to Perform a "Failure Report Correction" Task.

7.1.8 Conclusions and Lessons Learned

The prototype CEPA delivered at the end of the S phase is only a prototype of a portion of the software for a SEE and all tools are only prototype tools. The prototype CEPA provides an impression of what the software for a SEE should be like. "Look and feel" issues as well as functionality issues can be evaluated. The Cleanroom process is visible. A number of desirable functions for the software portion of a SEE are visible and available.

With only a modest amount of additional work the prototype can be used productively to support a "friendly" project.

However, considerable work will be required to develop CEPA into a production quality system to support substantial-sized software development projects. To accomplish this, the following tasks need be performed:

1. Enhance and use the prototype CEPA along with rudimentary tools to design software on a real project to learn about look and feel and required functionality.
2. Specify a complete CEPA.

The first specification should be for a CEPA implementation in the same style as the current implementation. Once that has been done, it may necessary to implement such a limited version to gain experience or it may be possible to directly proceed to specifying software that can customize itself.

3. Specify and implement a complete customizable CEPA.

There need to be two parts to a production CEPA. The first is the software that is used by people on a project after it has been customized for the process that has defined for the project. The second is the software that reads the process definition and then customizes the software so that it defines the project process. No design work has been done on the customizing portion of the software, so that some investigation should be performed to determine how to go about developing such software.

4. Develop versions of all the required tools to support Cleanroom engineering.
5. Adapt planning and scheduling tools to work within the CEPA environment.

The good news from the S15 project - CEPA work is that it is clear that it is possible to develop SEE software to support the Cleanroom process since it is a defined process.

The bad news is that CEPA definition work reinforces the proposition that a prerequisite to automating a process is that the process must be rigorously defined. The traditional trial-and-error method of software development is not and cannot be rigorously defined so it may not be possible to develop useful SEE software to support such processes.

7.2 Overview of the Process for Developing Process Applications in KI Shell

The knowledge acquisition and knowledge engineering tasks involved in defining selected aspects of process knowledge are fairly manageable. This is due, in part, to the common understanding of generally accepted notations to support the modeling of process knowledge. Chief among these notations is IDEF0 an activity modeling graphical notation language developed on a major U. S. Air Force program, the Manufacturing Technology (MANTECH) program. The IDEF0 notation is mandatory for use by participants of the Industry Modernization Incentive Programs (IMIP).

The basic IDEF0 notation is illustrated in Figure 37.

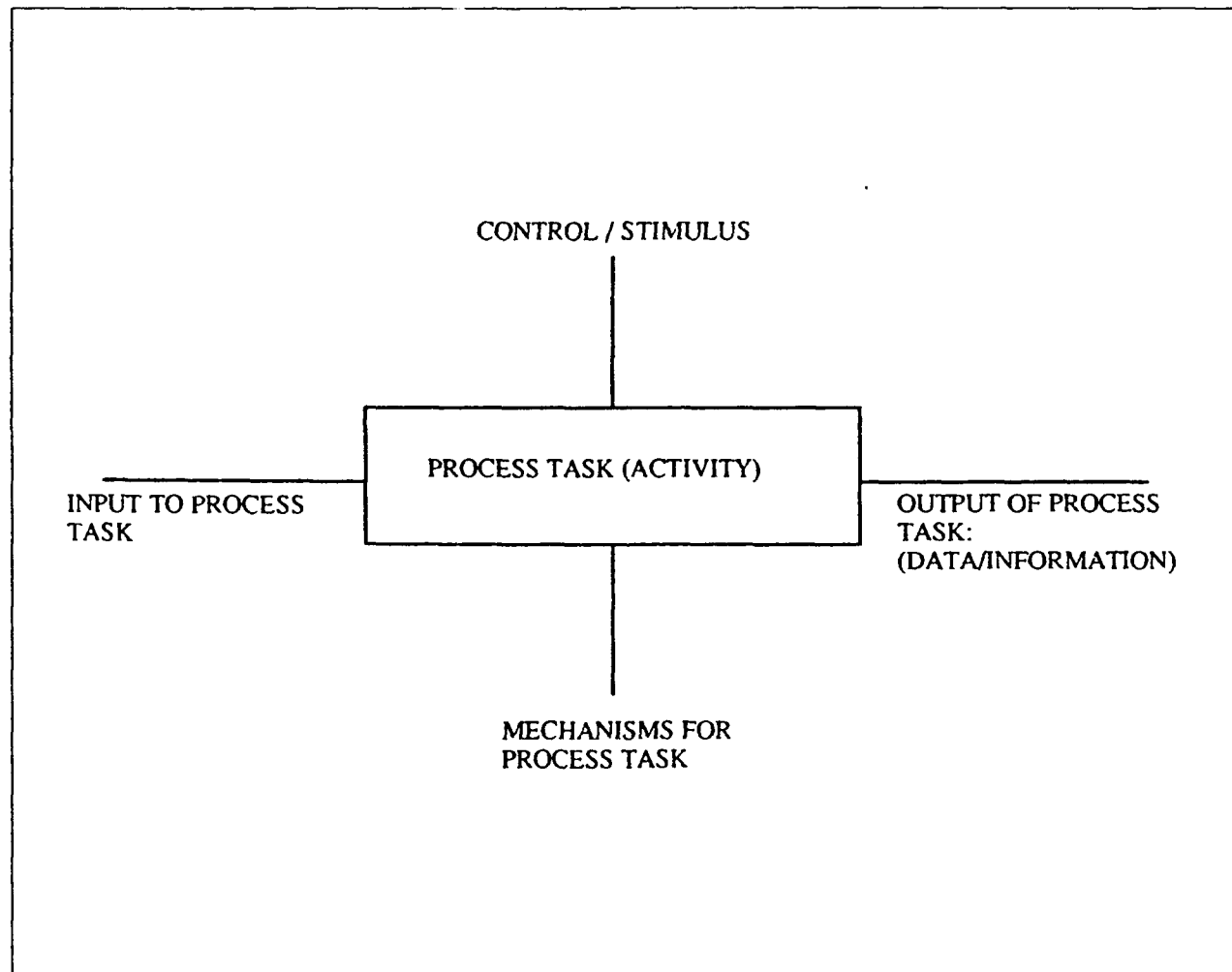


Figure 37. Form for an IDEF0 Process.

An IDEF0 activity, represents a process task in a process model. The input and output data flow represent data required by a process task and data or information produced as a result of the process task. The control or stimulus flow indicates signals or events that permit the process task to be performed. The mechanism flow identifies the mechanisms and tools required for performing the process task. Modeling notations, such as IDEF0, are widely available for use and provide a starting point for supporting process knowledge acquisition.

The development of process models using graphical notations such as IDEF0 addresses the development of process models at the "activity level," and as such, the development of activity threads for a project is referred to as "activity-based process modeling."

Given a specific model for representing the process knowledge acquired, for example, IDEF0, the next step is to transform the IDEF0 diagrams into a form that is enactable by a process enactment mechanism, such as KI Shell. In this process, numerous clarifications to the meaning of the process models developed will be required. To enact a model, an appropriate subset of activities (or process tasks) must be presented to the users, according to their project responsibilities. This is the concept underlying the role object in the KI Shell. A KI Shell method consists of roles, which in turn consist of a collection of activities to be performed by each role. Thus, the activity-based process models must be analyzed from the perspective of the agents who will be responsible for enacting selected processes. It is from this analysis that activities or process tasks are allocated to user roles. This is referred to as "role-based process modeling," and usually requires an activity-based process model as a precondition to performing role-based process modeling.

Another significant refinement to the activity-based process model would be specifying how inputs are transformed to the output of each process task.

The complete process for developing the software for enacting the role-based process model will now be described.

7.2.1 Process for Developing KI Shell Process System Applications

The KI Shell can be used to develop an executable *Process System*. By *process system* we mean the "system of processes" selected, defined, and designed to meet the process requirements of a software development project or a software development organization. A process system is developed by creating an architecture of processes to satisfy a project's requirements and preparing a design to implement the architecture. IDEF0 is a useful tool for modeling the processes required for a "system of processes" to support a project. Role-based process modeling is a useful activity for taking a set of processes and organizing them, based on user activities, to acquire sufficient detail to record them in a form suitable for enactment and presentation to process users. Thus, the result of implementing a system of processes to support the enactment of a project's software process, in an executable form, is referred to as an executable process system. We view the output of the processes codified in a KI Shell application as an executable process system.

The following definitions are useful in discussion below:

1. "AS-IS" process -- a system of processes that currently exist to support the development or production of some product, e.g., a software system or a computer system.
2. "TO-BE" process -- a desired system of processes needed to support the development or production of some product.
3. Process domain expert -- domain expert of a particular problem domain as well as the processes governing that problem domain.
4. Process engineer -- expert in acquiring, representing, developing, and implementing process models and enactable process systems.

The steps required to implement a KI Shell application are as follows:

1. STEP 1:

Entry: Initiate project.

Task: Create an "AS-IS" IDEF0 model of the current process. This is performed by the *Process Domain Expert*.

Validation: Team review and acceptance of the IDEF0 model.

Exit: Accurate IDEF0 "AS-IS" model.

2. STEP 2:

Entry: "AS-IS" process description.

Task:

- Analyze the "AS-IS" process and derive the "TO-BE" and desired process.
- Identify different "slices" or increments that could be implemented.

These tasks are to be performed by the *Process Domain Expert* in conjunction with the *Process Engineer*.

Validation: Team review of the "TO-BE" process.

Exit: Agreement of the potential "TO-BE" process increments that could be automated.

3. STEP 3:

Entry: "TO-BE" process increments.

Task: Perform cost/benefits analysis on potential process increments to be automated.

Validation: Team review of the cost/benefit and selection of increment to be implemented.

Exit: Agreement on "TO-BE" increment to be implemented.

4. STEP 4:

Entry: All process definition documents related to the method increment are made available to the process engineer.

Task:

- Present the KI Shell modeling concepts, tradeoffs, and user interface to domain experts.
- Jointly develop an initial method layout. This team task is to be performed by domain experts and knowledge engineers.

Validation: Initial methods reviewed. Participation of domain experts and process engineers.

Exit: Formal adoption of initial method layout.

5. STEP 5:

Entry: Detailed specifications provided by domain expert(s).

Task: More detailed process model developed by process engineers.

Validation: Formal walkthrough to establish common understanding of processes and how they will be enacted.

Exit: Formal adoption of initial increment to be implemented.

6. STEP 6:

Entry: Detailed process implementation construction plan prepared.

Task: Implement process increment.

Validation: Validate implementation by execution against process test cases, prepared from the defined process being implemented.

Exit: Accept increment.

7.3 CEPA Prototype System Development Implementation Log Overview

The purpose of this section is to provide an overview of the CEPA Prototype System development. The complete "CEPA Prototype System Development Implementation Log" will be provided as a supplement to this report.

7.3.1 March 23 through April 19

Entry:

1. All cleanroom documents made available.

Process:

1. Clarification of implementation team questions regarding Cleanroom.
2. Layout of initial method in KI Shell modeling notation (Figure 38 on page 91, Figure 39 on page 92).
3. Presented KI Shell notation to SET and IBM using Cleanroom Example from the "Cleanroom Engineering Software Development Process."

Validation:

1. Validated methods against the "Cleanroom Engineering Software Development process."

Exit:

1. Method layout informally reviewed by specification team.
2. Implementation team provided CEPA draft specification for CEPA prototype system.

7.3.2 April 22 through May 3

Entry:

1. Method layout informally reviewed by specification team.
2. Implementation team provided CEPA draft specification for CEPA prototype system.

Process:

1. Discussion of modeling tradeoffs by implementation team.
2. Implementation team members discussed development of a process-centered implementation strategy based on KI Shell concepts.

Validation:

1. Informal validation of implementation approach by demonstrating initial CEPA concepts to specification team.

Exit:

1. CEPA implementation approach informally validated.

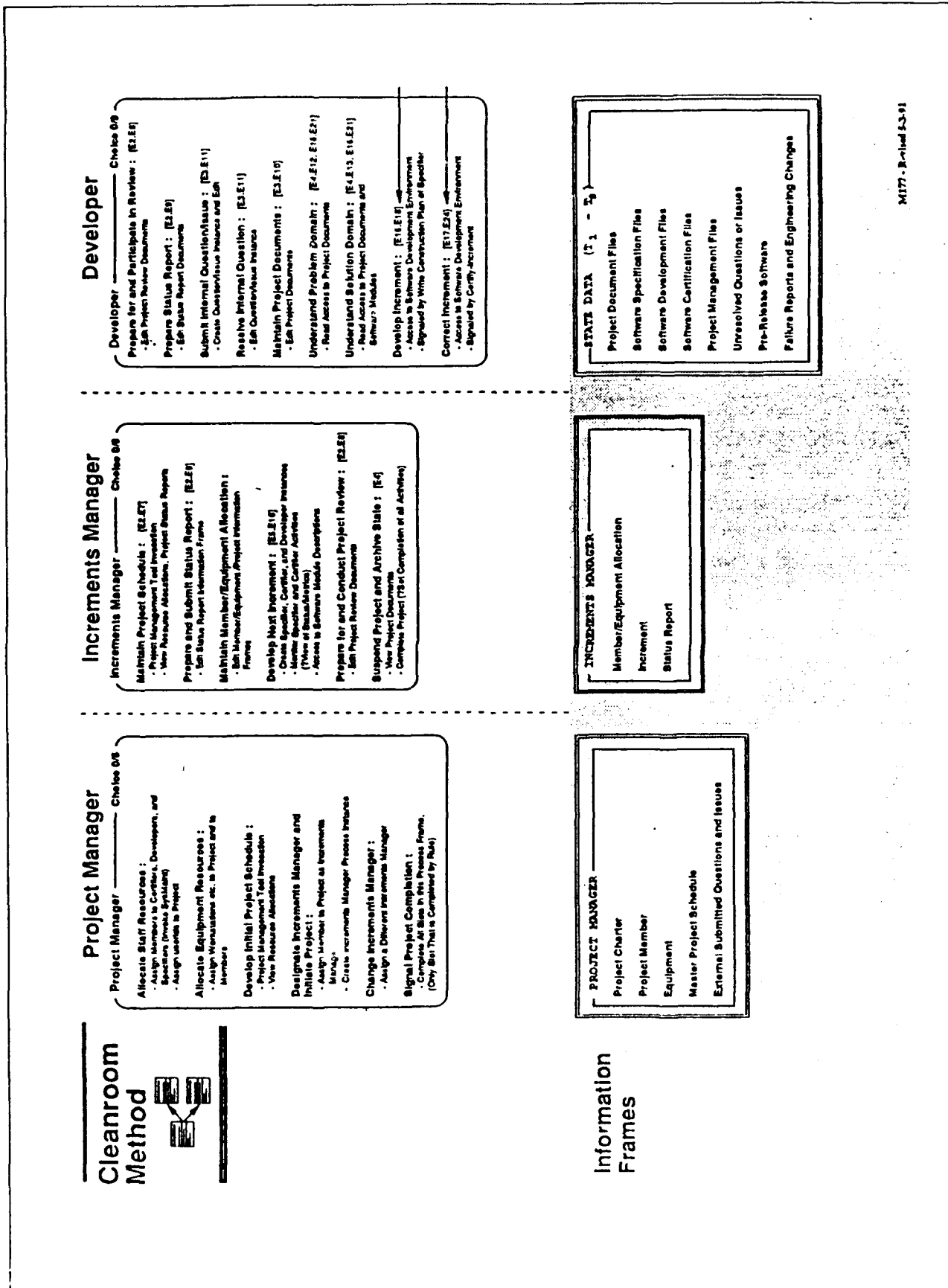


Figure 38. Cleanroom Process Method Layout (Part 1 of 2).

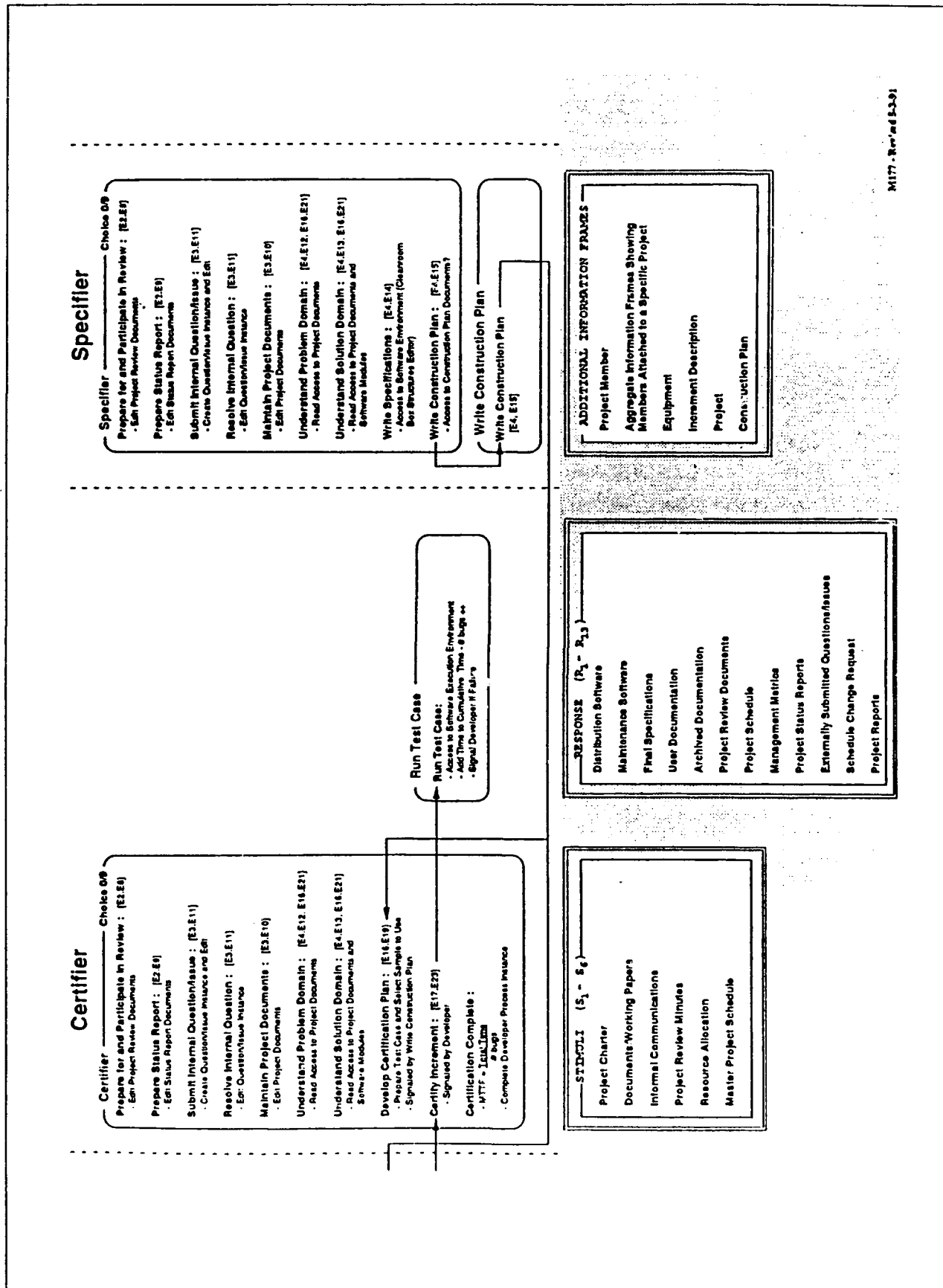


Figure 39. Cleanroom Process Method Layout (Part 2 of 2).

7.3.3 May 6 through May 24

Entry:

1. CEPA implementation approach informally validated.

Process:

1. Developed initial *Construction Plan* by implementation team.
2. Identified questions for specification team.
3. Studied CEPA specification, revision 1.

Validation:

1. Informal validation of CEPA implementation approach (i.e. mapping of CEPA specifications to KI Shell model) based on documentation.

Exit:

1. Review of Construction Plan by specification team.
2. Review of updated CEPA demonstration by specification team.

7.3.4 May 27 through June 12

Entry:

1. Review of Construction Plan by specification team.
2. Review of updated CEPA demonstration by specification team.

Process:

1. Execution of Construction Plan by implementation team.
2. Key features of CEPA prototype system implemented.

Validation:

1. Demonstration reviewed by specification team.

Exit:

1. Review of implementation approach completed.

7.3.5 June 12 through June 21

Entry:

1. Review of Construction Plan by specification team.
2. Review of updated CEPA demonstration by specification team.

Process:

1. Execution of Construction Plan by implementation team.
2. Key features of CEPA prototype system implemented.

Validation:

1. Demonstration reviewed by specification team.

Exit:

1. Review of implementation approach completed.

7.3.6 June 24 through June 28

Entry:

1. Review of implementation approach completed.

Process:

1. Presentation of the KI Shell User Interface implications of the process model to the implementation team.
2. Developed new CEPA screen regime. (Screens are available in the *"Cleanroom Engineering Process Assistant" specification.*)
3. Completed redesign of KI Shell approach to implementing the CEPA system by exploiting KI Shell mechanisms.
4. The redesigned approach to implementing the CEPA system was required to take better advantage of new KI Shell features and to improve the presentations of CEPA to the end-user.

Validation:

1. Review of redesigned method layout with specification team.

Exit:

1. Agreement by both parties on baseline method layout.

7.3.7 July 1 through July 30

Entry:

1. Agreement by both parties on baseline method layout.

Process:

1. Implementation of redesigned CEPA by the implementation team.

Validation:

1. Final CEPA validation testing by the specification team.

Exit:

1. CEPA ready for use to support *buoy problem* development.

7.3.8 CEPA Prototype Development Summary

The CEPA prototype was developed by using the spiral model, where each spiral increment had a set of goals and objectives and progress was reviewed against those objectives. At the end of each increment, the construction plan was reviewed against progress, and a plan for the increment was drafted. Increments were planned as two-week segments.

The CEPA prototype development is characterized in Table 2.

Prototype Number	Stage	Goal	Duration
1	Requirements Capture / Knowledge Engineering	Convey to implementation team KI Shell application development concepts and clarify requirements	4 days
2	Requirements Capture / Knowledge Engineering	Demonstrate Cleanroom process support, while clarifying aspects of the Cleanroom process, through prototype development	4 weeks
3	Interim Prototype Systems Development	Modify requirements engineering prototypes to conform with the CEPA specifications	2 weeks
4	Final Prototype Systems Development	Enhance interim prototype's man-machine interface to accommodate improved role screen design	5 weeks

Table 2. CEPA Prototype System Development Summary

7.4 Cleanroom Engineering Process Assistant Installation Instructions

In order to install the Cleanroom Engineering Process Assistant prototype system on an IBM Risc System/6000, you must follow the steps described in this section.

7.4.1 Pre-installation Activities

1. Check on the configuration of the target IBM Risc System/6000 on which CEPA is to be installed. CEPA requires:
 - a. AIX 3.1.5 (with the 3005 maintenance and updates applied)
 - b. AIXWindows
 - c. 16 megabytes of main memory
 - d. 600 megabytes of total disk storage
 - e. PTY devices should be set to 64, using SMIT
 - f. Number licensed users should be set to 3-32, using SMIT

2. Obtain a copy of the following software:

- a. Oracle RDBMS (Version 6 or higher), SQL*NET, and SQL*PLUS¹⁰ (available from Oracle Corporation)
- b. KI Shell runtime (available from UES, Incorporated)
- c. WordPerfect 5.0 for AIX (available from WordPerfect Corporation)

After the computer hardware configuration has been checked and the necessary software has been acquired, continue to the next step.

3. Create the following file systems from the "root" account:

- a. /u/oracle -- 60 to 90 megabytes, depending on Oracle products you wish to install, besides the basic CEPA application
- b. /u/wp50 -- 16 megabytes
- c. /u/kishell -- 86 megabytes
- d. /u/cepa -- 16 megabytes.

4. Establish the following computer accounts from the "root" account:

- a. oracle -- Oracle database product home account
- b. wp50 -- WordPerfect 5.0 product home account
- c. kishell -- KI Shell and CEPA product home account.

Make sure that the home directory for the above accounts refer to the file systems created. Further, make sure that these accounts employ the "C" shell (/bin/csh).

In creating these file systems, select the "automatic mount" option.

5. Mount the file systems that have been created

7.4.2 Install the Oracle RDBMS

An overview of the Oracle installation instructions are provided here. Please refer to your Oracle installation and planning guide to supplement the below instructions. The following instructions apply to Oracle version 6.0.31.0.1:

1. Logon onto the "root" account
2. From SMIT, create groups for "dba" and "oracle"
3. From SMIT, assign both the "root" and "oracle" accounts to the "dba" group
4. Change directory to /u/oracle and enter: `cd /u/oracle`
5. Enter pwd at the AIX prompt to verify
6. Mount the oracle tape and enter:
`installp -d /dev/rmtX all`
where X is the tape drive device number.
7. After the tape has been installed, reboot the IBM Risc System/6000 by entering:
`shutdown -rF`

¹⁰ If method recompilation is required, PRO*C and its libraries must be obtained.

8. Logon onto the root account
9. Set the path by entering:
`set path = ($path .)`
10. To execute oracle install enter:
`./oracle.install`
11. Make sure oracle home directory is set to /u/oracle
12. Accept the defaults for the logfile, oracle owner, local bin directory and install manual pages.
13. After the message "ORACLE BOOT install completed," reboot the IBM Risc System/6000 by entering:
`shutdown -rF`
14. After the reboot has completed, complete the installation instructions specified for "Completing the installation as 'oracle.'"
15. After ORACLE has successfully completed, edit the file "/etc/services" and add the following entry:
`orasrv 1525/tcp`
 Note: make sure the number is unique, and must be the same for all networked machines wishing to use an ORACLE database over the network.
`sqldba startup -- to startup the oracle database`
`tcpctl start -- to starts up the oracle SQL process`
16. Request all users to logout
17. Shutdown ORACLE, by entering the following commands:
`sqldba shutdown`
`tcpctl stop`
18. Reboot the IBM Risc System/6000.

7.4.3 Install the KI Shell / CEPA Files

1. Log on to account: kishell
2. Change directory to /u/kishell
3. Issue a pwd command to check on your present working directory
4. Insert KI Shell/CEPA tape into the tape drive and enter the following command:
`tar -xvf /dev/rmtX -- where X is the tape drive device number`
5. Copy/append the samples/cshrc.add to .cshrc
6. Edit the .cshrc file and complete the required values. The following is a completed .cshrc file:


```
# Replace the '___' appropriately and add to the end
# of the user account .cshrc
#
setenv ORACLE_HOME oracle      # Oracle installation directory
setenv ORACLE_SID CEG          # Oracle database instance is
setenv WP_BIN wp50/bin         # Word Perfect bin directory
setenv KI kishell              # KI Shell installation directory
setenv KI_BIN $KI/bin          # KI Shell bin directory
set path = ( . $path $WP_BIN $KI_BIN $ORACLE_HOME/bin )
setenv KIDBHOST T:ibmrs1:$ORACLE_SID
setenv KISYSDB kishell         # FileStore database location
setenv XENVIRONMENT $KI/Defaults/KiShell
setenv WPTerm gui_color
setenv WPTerm50 gui_color
set history=50
set filec=1
```

7. Create a subdirectory of /u/kishell called CleanRoom by entering:

```
mkdir /u/kishell/CleanRoom
```

8. Read in the "CleanRoom methods" tape¹¹ by inserting it in the tape drive, and entering:

```
tar -xvf /dev/rmtX -- where X is the tape drive device number
```

7.4.4 Create and Setup the CEPA Account

1. From the "root" account, create the "cepa" account. Please include the following during the setup of this account:

- a. home directory: /u/cepa
- b. shell: /bin/csh

2. Read in the diskettes entitled "SNAPSHOT.INITIAL" into the home directory of the "cepa" account by entering:

```
tar -xvf /dev/rfd0 -- where rfd0 is the disk drive
```

3. Make a directory called STATE in the /u/cepa directory by entering:

```
cd /u/cepa -- to change the directory
pwd -- to check the present working directory
mkdir STATE -- to make the state directory
```

4. From the root account, set all file and directory protections for /u/cepa to 777 by entering:

```
chmod -R 777 *
```

5. Edit (using vi or another editor) .cshrc and set ORACLE_SID, ORACLE_HOME and set the paths to the ORACLE, WordPerfect and KI-Shell executables.

6. Edit (using vi or another editor) the following files:

- a. RunSmall
- b. RunSmallFS
- c. RunPrint

¹¹ If source tape, read in method source code and associated makefiles.

d. RunPrintFS

Set KIDBHOST -- example: `setenv KIDBHOST T:ibirs1:$ORACLE_SID`

Set DISPLAY -- example:

`setenv DISPLAY unix:0` -- for IBM Risc System/6000 Console

`setenv DISPLAY xs1:0` -- where the xs1 is the hostname of the X-station

`setenv DISPLAY falcon:unix:0` -- where falcon is the hostname of the workstation (SUN)

7. Link CleanRoomX and CleanRoomXFS to the appropriate executables by entering:

`rm CleanRoomX`

`ln -s <CleanRoom method directory> /CleanRoomX .`

`rm CleanRoomXFS`

`ln -s <CleanRoom method directory> /CleanRoomXFS .`

8. Run the CEPA method using RunSmall, RunSmallFS, RunPrint and RunPrintFS. Project name (defined by CEPA Administrator) should be set to Buoy_System.

Note: If WordPerfect hangs up delete all WP related processes:

`ps -ef | grep wp` (gives you process ids)

`kill -9 <process id> , <process id> ...` (deletes the processes)

9. Copy the .cshrc file in /u/kishell to the /u/cepa directory by entering:

`cp /u/kishell/.cshrc /u/cepa/.cshrc`

7.4.5 Setup the CEPA FileStore Version

1. Create the following CEPA working accounts:

- a. sem -- systems engineering manager
- b. stl -- specification team leader
- c. dtl -- development team leader
- d. develop -- developer 1
- e. dev2 -- developer 2
- f. ctl -- certification team leader
- g. certifi -- certifier 1

Please include the following during the setup of this account:

- a. home directory: `/u/cepa`
- b. shell: `/bin/csh`

2. After the accounts have been established, send a mail message to each userid using the UNIX mail facility by entering:

`mail cepa sem stl dtl develop dev2 ctl certifi <enter>`

Subject: CEPA Accounts Established <enter>

Account established -- message ends. <enter>

CC: <enter>

3. At this point the FileStore version of the CEPA method has been installed and may be executed by logging onto the "cepa" account and entering:

xinit

RunSmallFS or RunPrintFS. (RunPrintFS is intended for screen printing.)

Note (1): Since the FileStore version is single-user, exit from each user before starting up the CEPA with the next user.

Note (2): To exit from X-Windows, close all windows in the X session. After all windows are closed, press the left mouse button and select the exit option. After completing this, press ALT-CNTL-BACKSPACE. This returns the workstation to command line mode.

7.4.6 Setup the CEPA ORACLE Version

1. Create ORACLE users named "kishell" and "CleanRoom" (case not important) by entering the following:

sqlplus system/manager

grant dba to kishell identified by sysmanager; < note semi-colons! >

grant resource, connect to cleanroom identified by cleanroom;

commit;

exit

2. Login onto the "cepa" account and import the ORACLE dump stored in Export/CleanRoom.dmp into ORACLE by entering:

imp cleanroom/cleanroom

When prompted for the import file name, enter:

Export/CleanRoom.dmp

Select the defaults for all other prompts.

3. Change directories to SysMaint and import the SysMaint database by entering:

cd Sys.Maint

import kishell sysmanager

(NOTE the difference between the two imports: The imp is the oracle import; the import is the KI Shell import.)

4. The ORACLE version of the CEPA should now be installed. Login as "cepa" and enter:

xinit

RunSmall (or RunPrint).

7.4.7 Archiving the CEPA Database

In order to make an archival backup copy of the CEPA database, the following procedure should be performed:

1. Open the CEPA window.
2. Change to the main CEPA directory for the CEPA account by entering:
`cd /u/cepa`
3. Obtain a dump of the oracle database by entering:
`Utils/OraExp`
4. Upon successful completion of step 3, change the directory to Export by entering:
`cd Export`
5. Rename the dated file to a suitable file name by entering:
`mv Sep_10_1991.dmp Work_in_prog1.dmp`
6. Change the directory to the main CEPA directory by entering:
`cd /u/cepa`
7. Format 4 diskettes, insert a diskette into the drive, and backup all of /u/cepa directory by entering:
`tar cvf /dev/rfd0 *`

7.4.8 Restoring an Archived CEPA Database

In order to restore an archived CEPA database, the following procedure should be performed:

1. Exit from the CEPA system before initiating a restore.
2. Log onto the "cepa" account.
3. Change the directory to the home CEPA directory by entering:
`cd /u/cepa`
4. Run the CEPA demonstration account housekeeping utility by entering:
`reload.exe`
5. Enter the SQL*PLUS utility to delete CEPA tables by entering:
`sqlplus CleanRoom/CleanRoom`
6. To delete the CEPA tables, enter the following SQL*PLUS COMMAND:
`SQL > start Utils/deltable.sqp`
7. To exit from SQL*PLUS enter:
`SQL > exit`
8. Restore files from tar formatted disks by entering:
`tar -xvf /dev/rfd0`
9. Import the restored CEPA data by entering:
`tar -xvf /dev/rfd0`
10. Import the restored CEPA data by entering:

imp CleanRoom/CleanRoom

11. You will be prompted to provide a filename for file import; Enter "Export/YourFile.dmp" at the prompt:

Import file: expdat.dmp > Export/YourFile.dmp

where YourFile.dmp is the name of the CEPA database dump file to be restored.

At the completion of the CEPA database restore, the message "Import terminated successfully" will be displayed. Your CEPA database has been restored.

7.5 CEPA Demonstration Operation Instructions and Script

7.5.1 CEPA Operation Instructions

To run CEPA, open an X-Window session by using the "xinit" command. Within the X-Window, log onto the appropriate cleanroom role by invoking "RunSmall." Instructions for initializing, archiving, and restoring the CEPA database are discussed in an earlier section.

7.5.2 CEPA Demonstration Script

It is desirable to use the CEPA prototype by showing it to software engineers and other people interested in learning about and studying software development process. Through interactions with these people, one can learn about the usefulness of software engineering environments.

The CEPA prototype is an engineering model of the integrating component of a Software Engineering Environment (SEE) that supports Cleanroom processes and engineering practices. The Cleanroom processes and engineering practices are built on a firm science base.

The purpose of the CEPA model is to:

1. Determine the feasibility of developing a useful SEE,
2. Develop requirements for a production-level product, and
3. Learn how to build and use such a product.

We know as a result of building a prototype CEPA, that a robust product version of CEPA can be developed. What we do not know is whether a CEPA is useful to practicing software engineers nor do we have real-world experience to develop requirements for a production-level CEPA. The first step in obtaining this information is to begin to demonstrate CEPA to interested people and obtain their reaction. CEPA demonstrations should be as open-ended as possible.

We believe that the CEPA model is the most advanced work yet done in the process world, either in the university or the research community. Therefore, we all have a great deal to learn by showing CEPA to people and soliciting their reactions.

7.5.3 CEPA Demonstration Script

The situation being demonstrated is three engineers (one development team leader and two developers) using three different workstations simultaneously in simulated time. But since there is only one person giving the demonstration (the demonstration manager) and one audience, the demonstration manager must move back and forth between roles at will to show the audience what is going on with each person. As a result, the script must leave a good deal to the discretion of the demonstration manager. He/she must be able to move back and forth between simulated roles as desired or as prompted by questions.

Preconditions:

1. Setup roles.
2. Setup state data with problem in some stage of completion with several boxes complete.
3. Adapt screen ready to go.

Demonstration situation:

1. One work station that will support 3 users. All screens for the three people are on one workstation screen so a person conducting the demonstration can transfer between roles asynchronously.
2. The duration for the demonstration will be between 15 and 30 minutes. That will be about the limit of the attention span during an exhibition. As one will see from looking at the demonstration script, extending it for longer periods of time will be easy.

Events for Developer One:

1. Works on design object. This assumes the actual editing of files in WordPerfect. Since working on any design object leads to a number of windows' being visible on the screen, the developer will work through some of them by designing a black box, a state box, or a clear box or refining or reorganizing a clear box.
2. Responds to mail from others.
3. Sends mail to schedule team review.
4. Holds team review.
5. Team review does not pass.
6. Does more work on the design object.
7. Submits a question.
8. Holds team review for the design object.
9. Team review passes.
10. Requests for team leader to circulate completion conditions.
11. Completion conditions are circulated and eventually signed off on. While this is occurring, the developer can go on to next task, which is a design object selected from the CEPA main menu.
12. Adds a note in Engineer's Notebook.
13. Looks at the Engineering Handbook to understand how to work on the subsequent design object.
14. Works on design object.
15. Looks at answer to question submitted earlier.
16. Calls up some state data to be viewed, to help create the design object.

17. Continues to work on design object.
18. Sends mail to schedule team review.
19. Holds team review.
20. Team review passes.
21. Requests for team leader to circulate completion conditions.
22. Completion conditions are circulated and eventually signed off on. While this is occurring the developer goes on to next task which is a design object selected from the CEPA main screen.

Events for Developer Two:

1. Works on design object. This assumes the actual editing of files in WordPerfect. Since working on any design object leads to a number of windows' being visible on the screen, the developer will work through some of them by designing a black box, a state box, or a clear box or refining or reorganizing a clear box.
2. Responds to mail from others.
3. Sends mail to schedule team review.
4. Holds team review.
5. Team review does not pass.
6. Does more work on the design object.
7. Resolves question when it arrives from developer one.
8. Holds team review for the design object.
9. Team review passes.
10. Requests for team leader to circulate completion conditions.
11. Completion conditions are circulated and eventually signed off on. While this is occurring, the developer can go on to the next task, which is a design object selected from the CEPA main menu.
12. Adds a note in Engineer's Notebook.
13. Looks at Engineering Handbook to understand how to work on the subsequent design object.
14. Works on design object.
15. Calls up some state data to be viewed, to help create the design object.
16. Continues to work on design object.
17. Sends mail to schedule team review.
18. Holds team review.
19. Team review passes.
20. Requests for team leader to circulate completion conditions.
21. Completion conditions are circulated and eventually signed off on. While this is occurring, the developer goes on to next task, which is a design object selected from the CEPA main menu.

Events for Developer Team Leader:

1. Assigns tasks.

2. Works on design object. This assumes the actual editing of files in WordPerfect. Since working on any design object leads to a number of windows' being visible on the screen, the developer will work through some of them by designing a black box, a state box, or a clear box or refining or reorganizing a clear box.
3. Sends mail to schedule a team review.
4. Holds team review.
5. Team review does not pass.
6. Does more work on the design object.
7. Responds to mail from others.
8. Looks at metrics.
9. Does more work on the design object.
10. Circulates completion conditions.
11. Team review passes.
12. Requests for team leader to circulate completion conditions.
13. Completion conditions are circulated and eventually signed off on. While this is occurring the developer can go on to next task, which is a design object selected from the CEPA main menu.
14. Looks at metrics.
15. Adds a note in Engineer's Notebook.
16. Looks at Engineering Handbook to understand how to work on the subsequent design object.
17. Works on design object.
18. Calls up some state data to be viewed, to help create the design object.
19. Continues to work on design object.
20. Sends mail to schedule team review.
21. Holds team review.
22. Team review passes.
23. Requests for team leader to circulate completion conditions.
24. Completion conditions are circulated and eventually signed off on. While this is occurring, the developer goes on to next task which is a design object selected from the CEPA main menu and proceeds to work on it. This assumes that the editing of files during the demonstration will be performed in WordPerfect. Since working on any design object leads to a number of windows' being visible on the screen, the developer should work through some of them.

Executing Demonstration:

All work is performed in the demonstration by pressing the appropriate CEPA buttons on screens and using the simulated Cleanroom tools built in Word Perfect.

7.6 Description of all CEPA Software Source Deliverables

7.6.1.1 CEPA Multi-User Source Code Files

The following files are necessary for preparing the ORACLE version of CEPA, which permits multi-user access to CEPA:

1. ProjectManager.c - Rules for the CEPA Administrator
2. SEM.c - Rules for the Software Engineering Manager
3. STL.c - Rules for the Specification Team Leader
4. Specifier.c - Rules for the Specification Team Member
5. DTL.c - Rules for the Development Team Leader
6. Developer.c - Rules for the Development Team Member
7. CTL.c - Rules for the Certification Team Leader
8. Certifier.c - Rules for the Certification Team Member
9. question.c - Rules to handle questions and issues
10. tasks.c - Code to handle the general Cleanroom tasks, e.g., Cleanroom Engineering Process handbook, engineering notebook, mail, view state data, refresh and logoff
11. Application.c - Invoking and terminating external applications integrated into the KI Shell environment, such as WordPerfect.
12. CRrolegraph.c - Graphical Cleanroom Display Manager (Adapt)
13. CRkinmetrics.c - Process metrics capture and reporting code
14. CRKey.c - Code used to generate unique identification keys for objects
15. Information.c - Rules attached to the information objects required for Cleanroom process implementation
16. Utilities.c - General purpose utilities written for the Cleanroom process implementation, e.g. circulation completion conditions, state data management, etc.
17. WorkAllocation.c - General purpose utilities to pre-allocate Cleanroom tasks.
18. attofid.sc - PRO*C routines for querying KI Shell objects based on their attributes. Call based on a single attribute.
19. attofidN.sc - PRO*C routines for querying KI Shell objects based on their attributes. Call based on use of multiple attributes.
20. kiuserX.c - File generated by the KI Shell method development environment.
21. dummyX.c - File for placing function call "code stubs" during software system development.

7.6.1.2 CEPA Single-User File Store Version Files

The following files are necessary for preparing a file store version of CEPA:

1. attofidNFS.c - Selects frame instances with certain attributes for the filestore version.
2. attofidFS.c - Selects frame instances with certain attributes for the filestore version.
3. CRKeyFS.c - Generates new identification keys for the filestore version

7.6.1.3 CEPA Make Files

The make file for building the ORACLE version of CEPA is:

Xmake

The command to execute make file for the ORACLE version of CEPA is:

make -f Xmake CleanRoomX

The make file for building the file store version of CEPA is:

Xmake

The command to execute the make file for the file store version of CEPA is:

make -f Xmake CleanRoomXFS

7.6.1.4 CEPA WordPerfect Files

Below are the list of files that appear on this diskette. These represent most of the files that are on the RISC/6000. The names are different on the RISC/6000 because of the fact that DOS only allows 8 character names with 3 character extensions. The files are organized into 3 main directories: init, broad and sensor. Each of these represent a different increment (thus a different module). The purpose of each type of file inside each directory are the same; only the information inside each one is different. Descriptions of each of the files follow:

1. VERSION.wp - contains the specifications for the entire buoy system.

The following files are in each directory:

1. STEP1_FI.WP - contains stimuli and responses of the module.
2. BLACKBOX.WP - contains the black box of the module.
3. STEP3_FI.WP - contains the black box validation of the module.
4. STEP4_FI.WP - lists the stimuli histories used in the black box.
5. STEP5_FI.WP - documents the decisions for state data distribution in the module.
6. STATEBOX.WP - contains the state box of the module.
7. STEP7_FI.WP - lists the usage of state data in the state box.
8. STEP8_FI.WP - contains the state box verification of the module.
9. STEP9_FI.WP - documents the more concrete data types for state data of the module.
10. CLEARBOX.WP - contains the clear box of the module.
11. STEP11_F.WP - contains the clear box verification of the module.

The following files appear only in the init directory, but will be created in the other directories as development and certification progresses:

1. TESTPLAN.WP - presents the test plan for the increment.
2. MARKOVMO.WP - presents the modified Markov model for the increment.
3. TESTSCRI.WP - presents the test scripts for the increment.

4. TESTSCEN.WP - presents the test scenarios for the increment.
5. EXPOUT.WP - presents the expected output of the test scenarios for the increment.
6. REFINEME.WP - presents a refinement of a clear box.
7. VERIFICA.WP - presents a verification of a refinement of a clear box.

7.7 Major Lessons Learned from CEPA Implementation

Cleanroom has well-defined activities and is rich in its synchronization and ordering requirements. It has several cases of dynamic work allocation and a complex information structure. Thus, Cleanroom is a suitable process to be supported by a KI Shell assistant, such as the CEPA prototype system.

To fully exploit KI Shell's enactment features at the lowest implementation cost possible, it is important to make precise, issues related to both the roles involved in developing a process system, such as CEPA and the KI Shell features necessary to support its development. It is critical that the following project goals be clearly understood by all process system development teams:

- Goal 1: Support the users of the process in the best way possible.
- Goal 2: Fully exploit the process enactment "shell's" capability to support the domain.

This requires good communication among the three roles involved in process system development, in the basis of an understanding of the team members' roles and responsibilities.

7.7.1 Process Implementation Roles

There are three types of roles involved in implementing a system to support the enactment of a process:

1. Process Domain Expert, who understands how the process is to be defined and presented to the users of the system.
2. Process Knowledge Engineer, who understands how the process enactment tool can be fully exploited to achieve the objectives of the process domain.
3. Process Implementer, who implements the process according to the specifications of the Process Knowledge Engineer that are presented in terms of the KI Shell notation.

7.7.2 Key Problem and Solution

In the task IS-15, the specification team's role was the Process Domain Expert, and the implementation team's role was that of Process Knowledge Engineer and Process Implementer. The specification team provided detailed CEPA specifications that described a concept for supporting Cleanroom. During the knowledge engineering and prototyping process, the interpretations of these specifications were those of the specification writers. To exploit the KI Shell's mechanisms for implementing CEPA, it was necessary to convey knowledge about the workings of the KI Shell to the specification team, so that the specifications could be presented to the implementation team in terms with which it was familiar. Thus specifications for a process system such as the "Cleanroom Engineering Process Assistant" would be easier for developers to interpret by specifying processing through the use of the KI Shell conceptual model. Therefore, the specification team and the implementation team require knowledge of the KI Shell conceptual model to effectively plan and design executable process systems.

During the initial process of planning the implementation of CEPA prototype, there was little communication between the specification team and the implementation team. This caused a situation where the implementation team had to abstract knowledge from the specifications based on one conceptual model to a

conceptual model with which the implementers were familiar. Although the box structure notation was excellent for conveying functional requirements, there was much room for extrapolating what the specifications meant with regard to man-machine interface and display regimes. For example, the CEPA specification called for the following:

1. If there were no tasks of a certain type for a user (such as develop state box), that task option should be disabled
2. If there were tasks, the user should be alerted in some manner. (the user should not have to issue a command to find out if there are any pending tasks).

The standard KI Shell menu system did not include the functionality to support these requirements. Further, the implementation team also had to interpret navigation logic implied in the CEPA specification.

It would have been useful for the specification team's domain experts to have understood, in a more precise way, the KI Shell's presentation and user interface implications of the underlying process model.

Most of the above problems were remedied through an intensive two-day team communication session where the specification team was given better knowledge of the KI Shell conceptual model and the implementation team was able to explore beyond their traditional methods for implementing process system applications by incorporating new techniques. The major lesson learned here is that to develop specifications for complex process systems such as the *Cleanroom Engineering Software Development Process*, either one of two models must be employed:

1. Knowledge acquisition through the interviewing of domain experts, the incremental representation and validation of process knowledge, and the incremental implementation of the process system
(Using this method, both domain expert and knowledge engineer build conceptual models of the domain and the application of the tools being employed. Where there is no shared model for how knowledge is to be represented and employed, this is the only practical model to select.)
2. Knowledge acquisition through the analysis of prepared materials and specifications, incremental representation and validation of process knowledge with best available personnel or domain experts, if available, and the incremental implementation of the process system.
(Using this model, it is vitally important that the specification team have knowledge of the conceptual model of the implementation technology that is to be employed. If specifier and implementer share the same understanding of knowledge representation and implementation models, the functional specifications and man-machine concepts have a better chance of being understood by the implementers. Further, it is important for the specification team and the implementation team to have access to one another, both during the specification preparation process and during the specification analysis and design process, to help interpret the specifications.)

7.7.3 KI Shell's Suitability for Cleanroom

For the practices of an enterprise to be suited for support using a KI Shell-based assistant, the processes that underlie these practices must:

1. Be well-defined, i.e., consisting of well-defined activities;
2. Have some precedence in the order in which they must be executed or require some synchronization among them;
3. Do some degree of dynamic work-allocation (i.e., based on the results of some previous activity, create work for some user); and
4. Require the modeling and viewing of structured information.

The processes underlying the Cleanroom methodology do meet the above requirements, and hence are suitable for representation using the KI Shell. For example, the activities in Cleanroom consist mainly of:

- Define User
- Define Project
- Circulate Completion Lists
- Sign Completion Conditions
- Receive Completion Conditions
- Allocate Personnel
- Allocate Teams
- Develop Black, State, or Clear Box or Refine a Clear Box
- Run Test Cases.

All these activities are well defined, and as such were suitable for implementation (that is, they can be implemented algorithmically).

Cleanroom activities must be done in a certain order. For example, most certification activities (such as the running of test cases) must be done only after development of an increment is complete. There are several examples of such ordering requirements in the Cleanroom methodology.

Work allocation is a primary activity within Cleanroom. For example, Certifiers create failure reports, which then become pieces of work to be done by the Developers, who then have to correct the code appropriately.

The information to be processed in Cleanroom has a complex structure. While black, state, and clear boxes are just treated as text files, they must be grouped within modules, within projects, and so on.

Cleanroom is a suitable application to be supported by the KI Shell, as it (1) has well-defined activities, (2) is rich in its synchronization and ordering requirements, and (3) has several cases of dynamic work-allocation and a complex information structure.

8.0 STARS IS-15 Software Representation Work

On the basis of the Software Process Management System (SPMS) prototype work performed under STARS Task IR-23/B, IBM decided to redirect work from performing software process modeling experiments using SPMS to examining process management architecture issues for the IBM STARS SEE and to examine how the Software Process Management System prototype could be migrated to the IBM STARS SEE. Considerable process modeling has been performed in SPMS, including the process asset capture of the IEEE P10-74 software life-cycle process components. Codifying these process assets provided the basis for experimenting with instantiating process architectures through a reuse library of process assets.

8.1 Software Process Modeling Support

This section describes an example of a system intended to provide support for the modeling of software processes, namely the Software Process Management System (SPMS). We shall provide an overview of SPMS, the SPMS concept of process modeling, and give a description of the features of SPMS including its software process simulation capabilities. We shall first assess the requirements for the port of SPMS to the IBM RISC System/6000, then discuss of the training materials developed, and finally consider the hardware/software requirements of SPMS.

8.1.1 Software Process Management System: Overview

The prototype SPMS supports exploration and experimentation concerning some of the issues introduced in section 2, such as defining software processes, software process enactment, process improvement and product metrics. We believe that it will facilitate communication of a formalized process across an organization or project. It will also support some degree of *process reuse* by allowing selection of a process model from a base set of alternatives. It will also support process evolution and adaptation owing to its ability to define model-based product and process measurements, and to collect and reason about data relative to the process model.

The prototype SPMS meets some of the system requirements for enactment described in section 2.2.3.3. SPMS supports to some degree most of the process model concepts needed for enactment including products, activities, agents, control flow, communication, decisions, long-term execution, concurrency and communication, views, and to some degree roles, extensibility, reuse, process change, and the testing and debugging of process enactments. The prototype SPMS does not support all of these aspects to the degree necessary to meet the requirements found in section 2.2.3.3. It has, however, allowed experimentation and refinement of requirements as well as the development of techniques that we believe can be utilized to fulfill these requirements.

The SPMS concept of software process management takes an activity-based view to modeling process that is related to key concepts of project planning. Figure 40 on page 112 illustrates the SPMS software process management concept.

Process Model = How

Project Data = What

Plan = How + What

Resources = Who

Durations + Scheduling = When

Scheduled Plan with Resources = Who, What, When, How

This Plan + monitoring methods + enaction =

Software Process Management

Figure 40. What Is Software Process Management?

8.1.1.1 SPMS Architecture

A high-level view of the SPMS system architecture is shown in Figure 41 on page 113. This figure illustrates SPMS as an element of the software development project software engineering environment along with directly related project disciplines. SPMS can be viewed as a tool for interfacing among all project disciplines and corporate overhead functions, by including these organizational interfaces in the process models developed and tailored for a project.

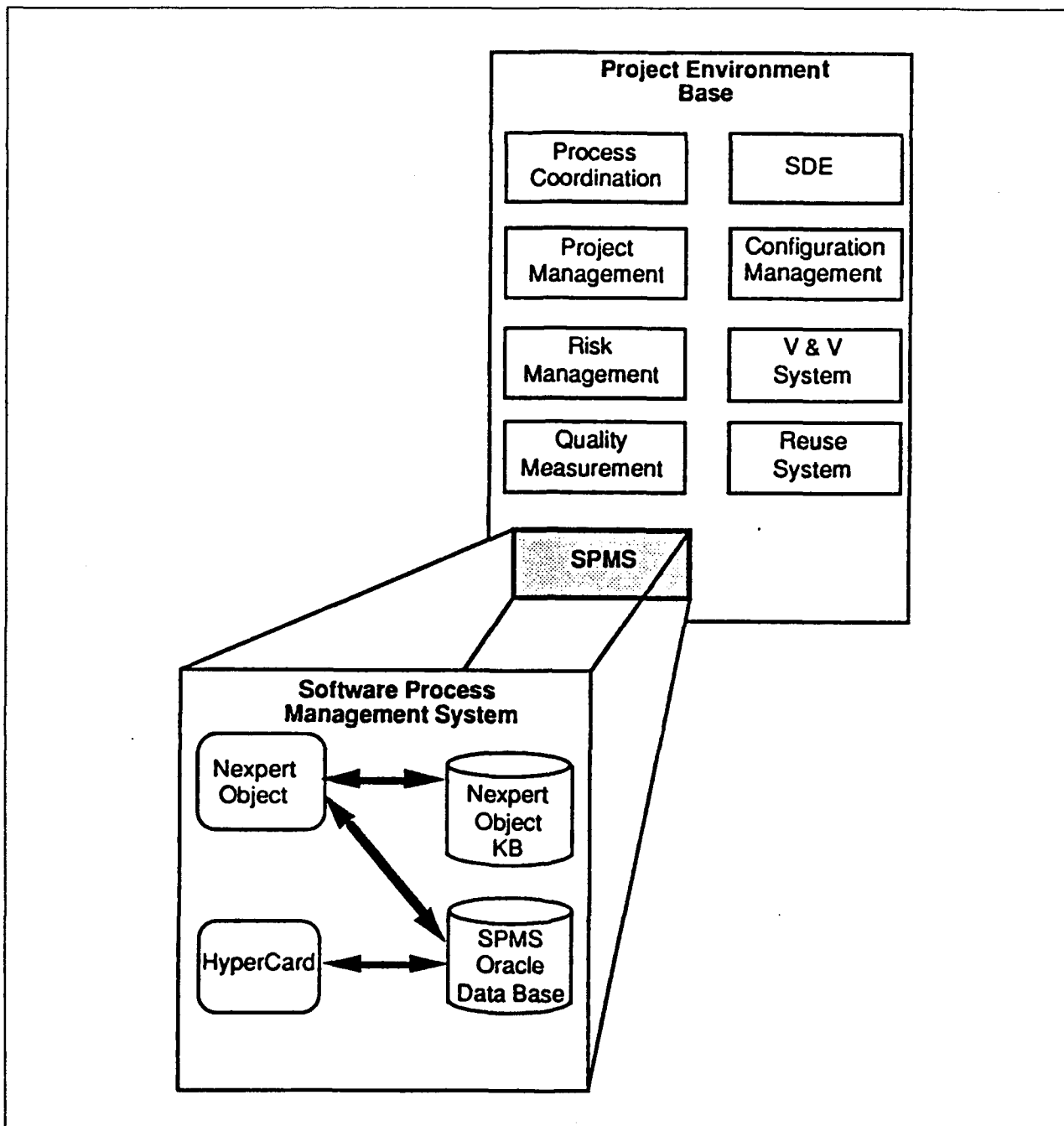


Figure 41. High-Level Architecture of SPMS.

The intended users of SPMS are project managers, process engineers, and software engineers. SPMS provides a means of collaboration among these users in a software development project. The determination of which components of the SEE should directly interact with SPMS was based upon the SPMS concept for modeling processes and model- and project-specific tailoring. The internal architecture of the SPMS prototype that was developed on IR-23/B is illustrated in Figure 42 on page 114. The specific COTS tools being used in the SPMS prototype are described in section 8.2 of this document.

The system architecture for SPMS is comprised of the following system components:

1. A Control Integration Mechanism

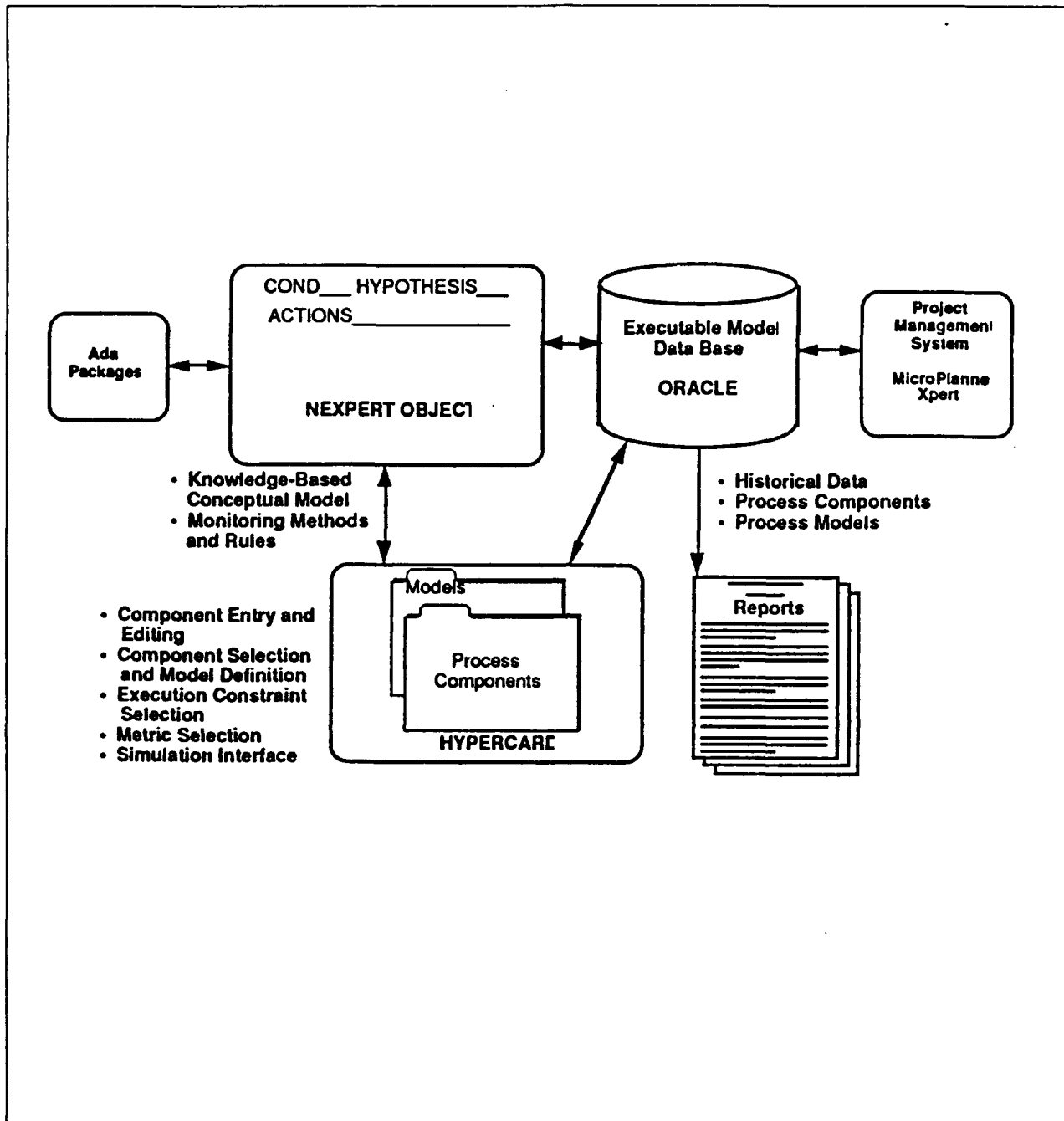


Figure 42. The System Architecture for SPMS.

HyperCard provides control integration services for integrating the commercial-off-the-shelf tools for SPMS. The HyperCard interface is directly connected to the database for the purposes of entering, browsing, and selecting the components, constraints, or metrics for building a process model. The HyperCard also serves as the interface for the system when it is used for simulation.

2. Classification Knowledge Modeling Tool and Method Monitoring and Execution

The expert system shell (NEXPERT Object) contains the knowledge-based representation of the process model and the instantiated project-specific model. It also contains the methods used for monitoring the executing process.

3. Persistent Knowledge Store

The ORACLE relational database contains the process components library, the process models that have been constructed, the project-specific model, and historical data. ORACLE's role is also that of a persistent data storage mechanism to support process enactment simulation, by maintaining process state and history.

The arrows to and from the NEXPERT Object expert system shell represent bridges between the shell and the other COTS tools that we have used in SPMS. These bridges allow the expert system shell to be embedded into SPMS within the Project Environment Base prototype (software engineering environment). The shell may call out to procedural languages such as C or call out to the relational database, using SQL to retrieve from or update the database. The bridge to HyperCard allows SPMS to directly interact with the NEXPERT Object expert system shell.

NEXPERT Object provides a rich classification knowledge modeling tool, and supports object-oriented systems development that provides full multiple inheritance. By using the NEXPERT Object product in conjunction with the ORACLE product, we have not only many of the advantages of an object-oriented database but also a more mature and widely available technology.

4. Project Activity Network Modeling Tool

The MicroPlanner Xpert provides functionality common to many project management systems, such as PERT and Gantt charts. Further, MicroPlanner Xpert is employed to perform process task or activity modeling. The activity networks are exported for use by SPMS for developing the "project/process" plan -- the combining of process activities with the details necessary to develop effective process models. The "project/process plan" concept is illustrated in Figure 43 on page 116.

8.1.1.2 SPMS Process Modeling Concepts

Process models and their representations must be viewed at several levels of granularity. Among these are the entire model, named groups of tasks and their relationships, and the individual tasks themselves. To further complicate the issue, each of these levels of granularity may be mapped to different levels of conceptual abstraction, depending upon the process to be modeled, the state of knowledge concerning the various tasks within it, and the need to specify different tasks at different granularities.

SPMS is designed to be independent of the particular level of conceptual abstraction and allows the user to specify process fragments or nodes within named groups of related process fragments and to relate these named groups or process components into a larger model. The named process components may optionally be deleted from the larger collection or library of components if desired. For example, some of the named components might represent different techniques for producing the same products. Each of the techniques contained in a named process component is related to those process fragments or nodes within other named components that require their products as inputs. If some of these techniques were not ever of interest, then the user might delete these from the process component library or model by using the model editor of SPMS. Other techniques might be of interest and left in the model, but the user might not wish to specify exactly which technique was to be used on a particular product until project-specific information was known. Parameters associated with the various techniques allows a late binding of project-specific data to the more general model.

8.1.2 Software Process Management: Concept of Operation

Any software process management capability should support the definition, design, and continual development of software processes. It is these models that will be used as "templates" for further elaborated processes for a specific project. SPMS provides the facilities for developing and describing these generic models and instantiating them for a particular project. The instantiation includes time and resource scheduling. SPMS also provides the mechanism for simulating these processes and replanning based on input from the user.

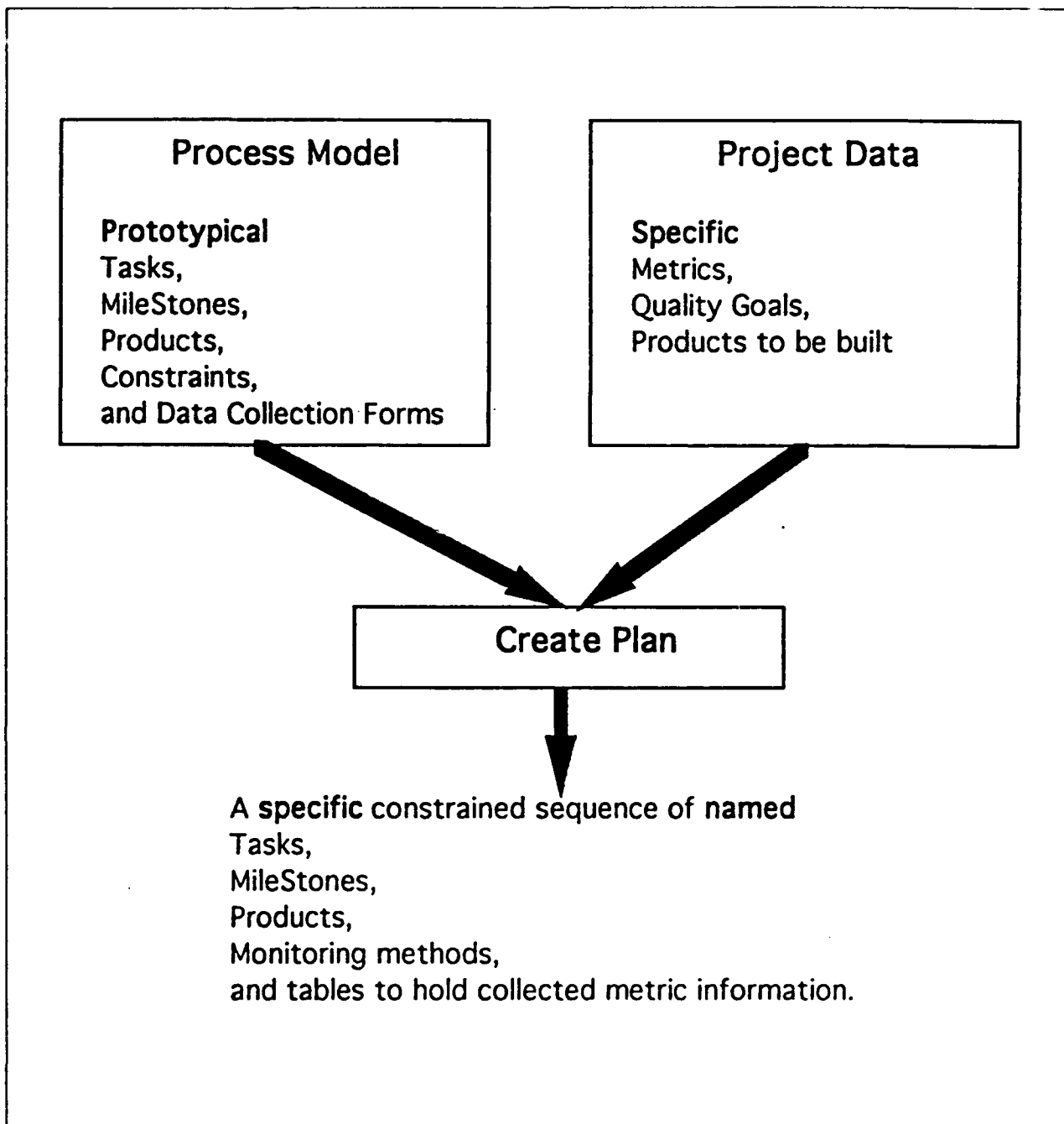


Figure 43. Project/Process Plan Concept. Process Model + Project Data = "Project/Process Plan."

Figure 45 on page 119 shows a high-level view of the operation of SPMS. Starting at the far-left side of Figure 45 (point 1), process and product components and constraints are created to form the basic building blocks of the process model. These user actions are performed within the project management system, MicroPlanner Xpert (Xpert), which results in a graphical representation of the process model. Activities (or process tasks) are modeled in Xpert by using an Entry / Task / Validation / Exit (ETVX) style. The form for an example SPMS process model component is illustrated in Figure 44 on page 117.

The metrics to be collected and the data collection forms to be used in collecting this data have been implemented in the prototype SPMS from the "RADC Quality Framework (Technical Report (Interim) Volume IV Software Quality Framework," <40> (point 2) and are contained within a metrics database. This database and the associated RADC documents provide the information necessary to select data collection forms

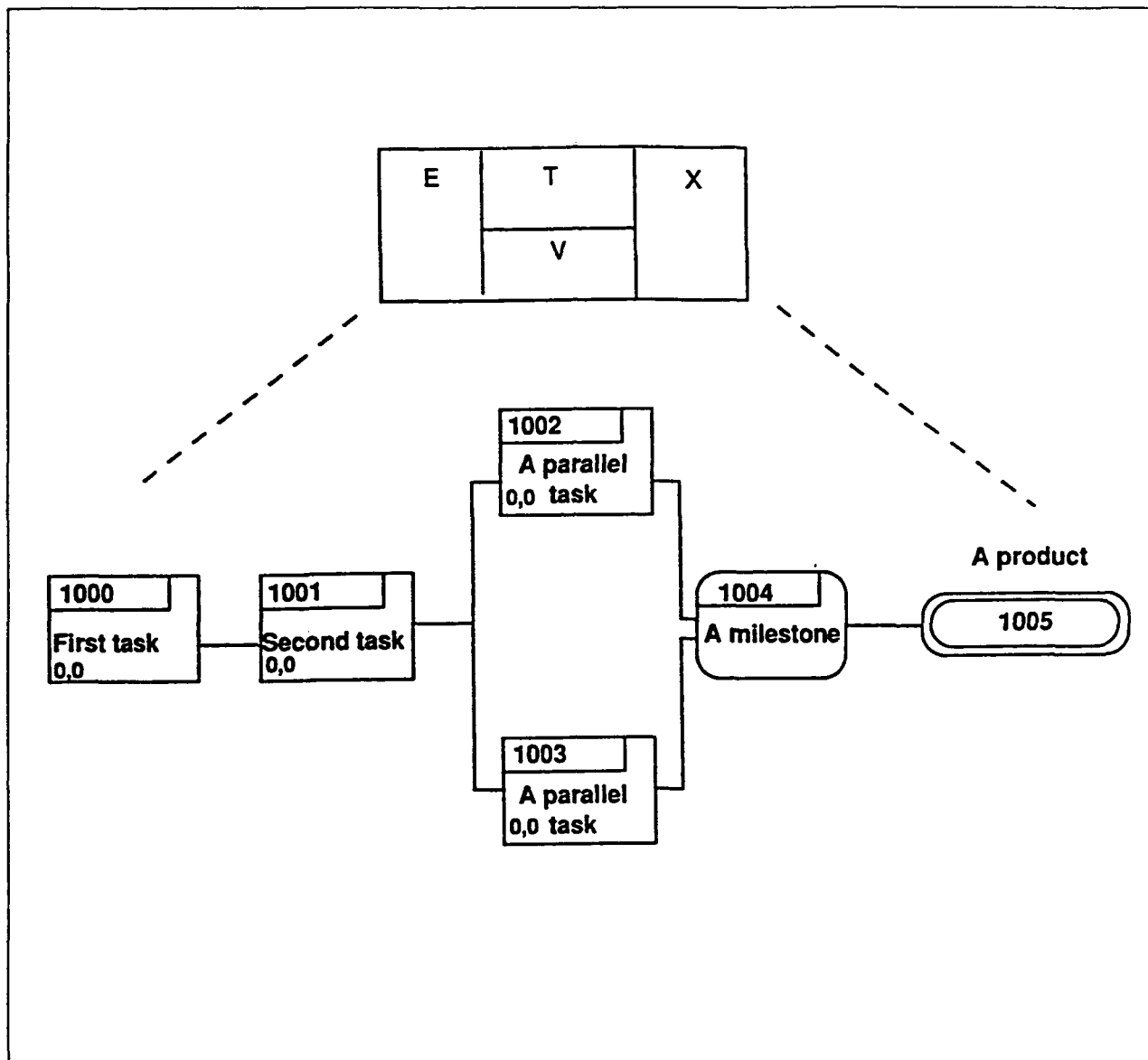


Figure 44. A Process Model Component.

(DCF) to be associated with validation tasks within the process model. (See point 3.) A node may also be specified in the process model as a starting point for potential rework should the measures taken by the validation task fail to meet desired quality goals.

The process model is then exported as an ASCII file from the project management system. (See point 4.) The process engineer (PE) may exit the project management system.

The PE logs into SPMS and, from the process interface, imports the ASCII representation of the process model into SPMS. This constrained and instrumented process model may be used as the basis for creating project-specific plans (see point 7) or used as a reusable library of tailorable process components. Tailoring may occur at two levels of granularity:

- The individual process, product, and constraint components (see point 5)
- The model level (see point 6).

The edited model may then be used to create plans.

Within the Project Interface, a project can be created (see point 7) within SPMS. A project represents the combination of quality measures and software components to be constructed by the software development process. The quality measures and their associated quality goals are selected at point 8. The software parts are also associated with this project at point 8. When the project has been constructed, a plan may be created (see point 9). Creating a plan combines the process model information with the project information to produce the instantiated plan, knowledge bases, metric calculation methods, and tables necessary for the execution of the plan. (The plan cannot be executed before scheduling by the project management system.)

The PE may tailor the plan by editing the quality goals for specific products within the software development process. The quality goals selected at point 8 provide default values for a selected software quality factor for all instances of products measured by that factor. The tailoring of these values at point 10 allows the setting of higher- or lower-quality goals for specific products within the plan.

The plan may have specific graphical displays associated with it. (See point 11.) The selected graphs will be displayed and dynamically updated during the simulated execution of the plan. The database is kept updated by the expert system during execution, and the graphical displays are updated once for each increment of the simulation clock. Before simulated execution, the plan must be exported to the project management system for scheduling. An ASCII file is exported from SPMS in the format required by the project management system, Xpert. (See point 12.)

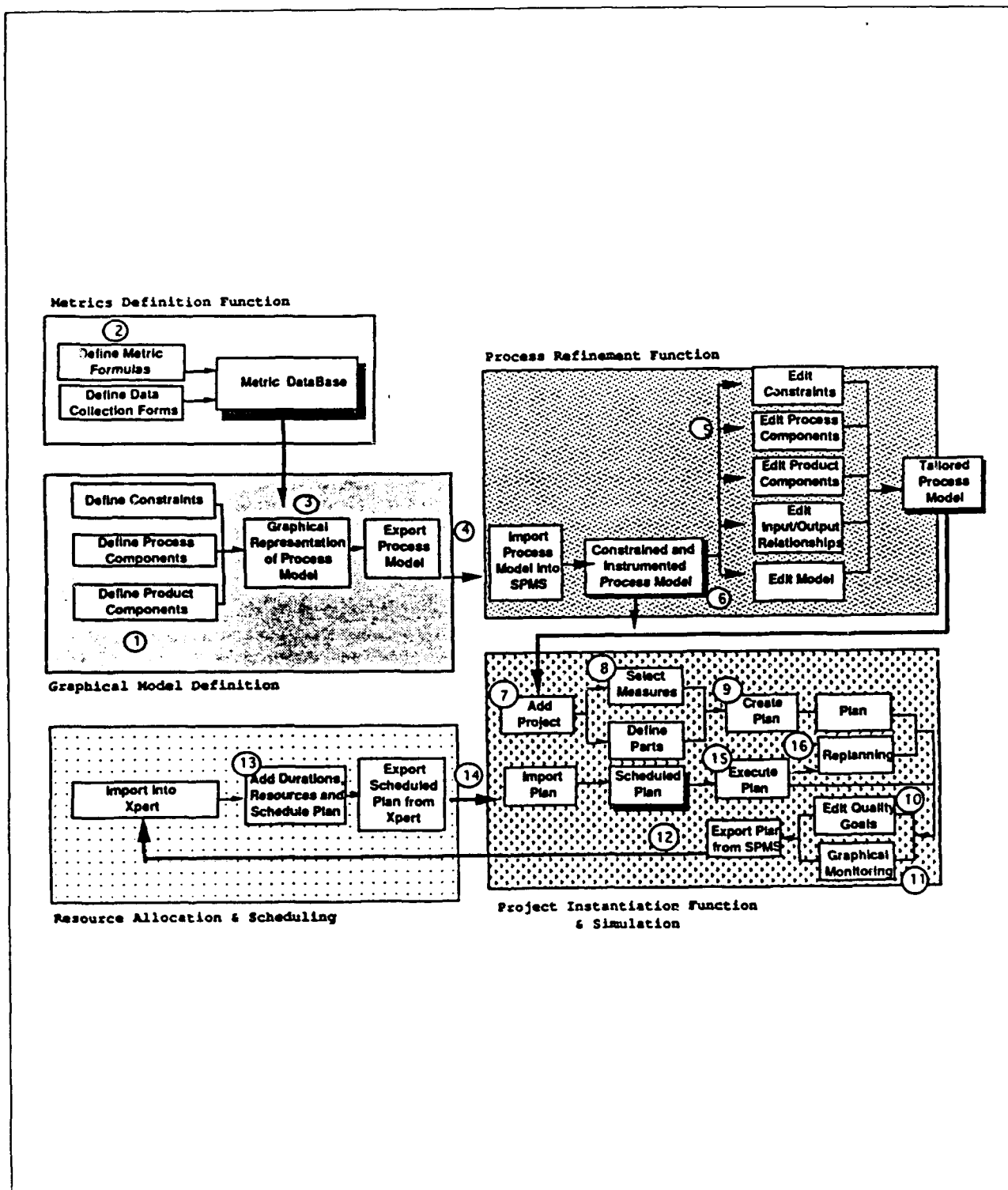


Figure 45. A High-Level View of Operation of SPMS.

This unscheduled plan is imported into the project management system (Xpert), where the appropriate duration estimates for each task must be entered, the desired resources must be associated, and the desired plan must be scheduled. When these actions have been performed, the scheduled plan is exported from the project management system (Xpert) as an ASCII file (see point 14).

The scheduled plan must be imported into SPMS through the Project Interface. At this point, execution of this plan may be simulated. (See point 15.) The metric monitoring methods within the expert system will request that the necessary data collection forms (DCF's) be completed as they are needed within the executing plan. In the simulation, the values of the DCF questions will be randomly computed. The monitoring methods will then compare the computed values of the metrics (which are based on the answers to the DCF questions) with the quality goals for the quality factor for the specific product being tested. If these values are within the range of the desired goals, then execution of the task sequences that follow the validation task in the plan will continue, otherwise they cannot continue, and failure notification is issued.

If desired, those portions of the network that the process model has specified as necessary to alleviate the failure of the validation task may be automatically created. This represents the "rework" needed to bring the product that has failed the validation task up to the level of the quality threshold. (See point 16.) This replanning is based upon a rework path that was specified within the process model. The impacts of new product versions that may result from rework are also propagated through the network.

As a result of replanning, it is necessary to reschedule the network of tasks. This requires moving through points 12, 13, and 14 before continuing with the execution of the plan.

8.1.2.1 Software Process Management System: Process Modeling Concept

Because the prototype SPMS utilizes generic process models to produce project-specific process models (plans), it is important to clearly distinguish between the two kinds of process models. A *process model* does not represent any specific software component or system; it is the sequence of tasks, milestones, constraints, and products necessary to produce a prototypical single instance of each of the types of software components that are represented within the model. This is in contrast to an *instantiated project-specific process model*, which usually contains numerous specific named and interrelated instances of the software components to be produced by a software development process. This is usually called a *plan*.

A plan may have resources, durations, scheduled start and finish dates, cost information, and work breakdown structures associated with the named tasks and products within the plan. Process models normally lack this detailed elaboration but contain information concerning the development mode of the tasks, the architectural level of the products that the tasks produce, the data collection requirements of the metrics associated with particular tasks, and starting points for potential rework within the model.

Process models are used to provide the framework for producing plans that may be replicated. Process models may be viewed as a collection of process components or process model fragments. The process model contains these components or fragments and their relationships. Some writers refer to the components as the process model and the view which includes the relationships between these components as process architecture. It is important to differentiate between non-specific process models which represent prototypical instances of tasks and products, and instantiated ones which represent an actual software development process.

A *plan* is the baseline for monitoring progress of a specific software development project. The tasks, milestones, and products within both process models and plans are represented within an activity or node network in SPMS. This network is entered into the system via traditional project management techniques.

The node types used to support process modeling, that are currently supported in the prototype SPMS are:

- **Task:**

The basic component of an activity or node process model. It signifies that something is going to happen.

- **Milestone:**

A milestone is a special node used to highlight important events in a process model.

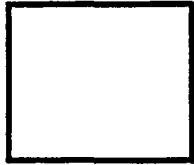
- **Interface:**

An interface node is a special node used to provide a logical link between two or more named groups of process components. An interface node is used to represent a product in SPMS.

- **Reverse (Or):**

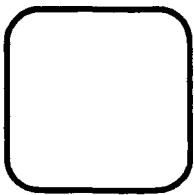
A reverse node is a special node that allows a task to start as soon as any of its predecessors is complete as opposed to the normal logic in which an operation may only start when all of its predecessors are complete. It represents an OR condition rather than the usual AND condition in SPMS.

These symbols are illustrated in Figure 46 on page 122.



- **Task:**

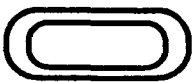
The basic component of a process model.



- **Milestone:**

A special node used to highlight important events in a process model.

- **Interface:**



A special node used to link subnetworks in the process model. Represents a product in the SPMS.

- **Reverse: (OR)**



A special node used which allows the successor to start when *any* predecessor is complete rather than when *all* predecessors are complete.

Figure 46. Process Activity Modeling Symbols.

These nodes may be linked into an activity network by the following constraint types:

- **Finish to Start:**

A Finish to Start link is the most common link. It specifies that a task cannot start until its predecessor is completed.

- **Finish to Finish:**

A Finish to Finish link signifies that the completion of a task is in part determined by the completion of its successor task.

- **Start to Finish:**

A Start to Finish link specifies that a task cannot finish until its predecessor starts.

- **Start to Start:**

A Start to Start link specifies that two tasks may start together.

- **Hammock:**

A Hammock link calculates its own duration as the elapsed time between its start node and its ending node. Hammocks may have resources and are used to summarize parts of a process model network.

These symbols are illustrated in Figure 47 on page 124.

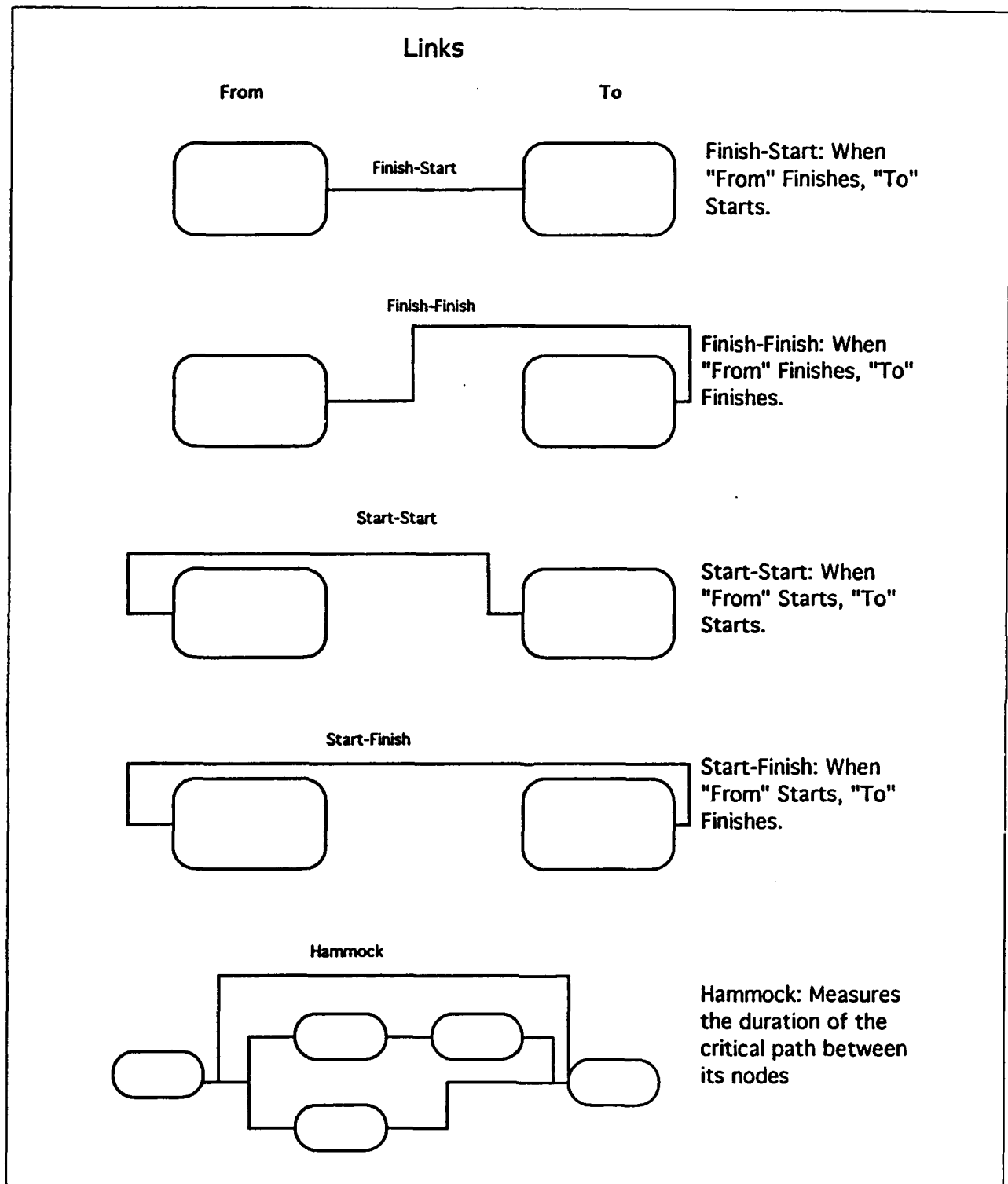
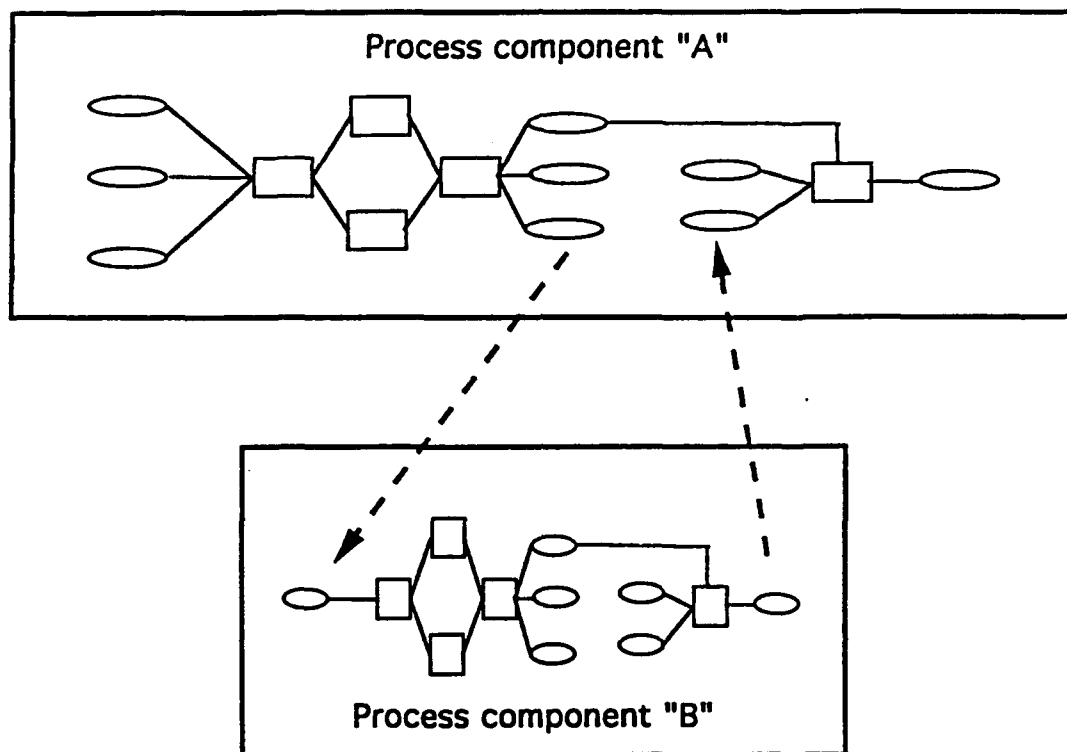


Figure 47. Process Activity Links.

Interface nodes are used to represent products in SPMS. Hammock links connect nodes that represent the starting point and finishing point of some development phase. The duration of this link is the sum of durations of all links and nodes between the two points. As such, Hammock links are used to represent development phases. The activity network may include coarse- and fine-grained process components within the same model. The coarse-grained components are named subnetworks of activities. These may be hierarchically ordered within SPMS. The fine-grained components are specified within the coarse-grained components and

consist of the individual elements of the activity subnetwork. Both the coarse- and fine-grained process components may be edited as discussed in section 7.1.2. Figure 48 on page 125 illustrates an example of process expansion in a SPMS activity model, where the details of the aspect of process are describes at a finer grain. The subnetworks are linked via interface nodes (products in SPMS) among the coarse-grained components to form the complete process model.



Some Possibilities:

Process component "A" might be at the SYSTEM level.

Process component "B" might be at the CSCI level.

"B" might be considered "Part of" "A"

Process components "A" and "B" might be at the same level and "B" specify greater detail than other portions of "A". "B" might be considered "Part of" "A"

How do you want to view parts in the model editor?

Figure 48. Process Granularity / Expansion Concept. This figure illustrates an example of process expansion in a SPMS activity model, where "process component A" and "process component B" are at the same architectural level, e.g., CSCI level. Process "B" might be a finer grained expansion of a process represented in "process component A."

The process components are parameterized to indicate the development mode and architectural granularity of the particular component. For example, a process component may be specific to a sequence of prototyping or reuse activities. This same component might operate on a subsystem (CSCI) level software compo-

ment. Each process component within the model contains values for these parameters. The user provides project-specific data indicating the name, architectural level, development mode, and part of relationship of a software component to be built by the software development process to the SPMS. This project-specific information is used to determine which portions of the parameterized process model must be instantiated into a project-specific plan.

Process models in SPMS may also include references to a process measurement model. The measurement model is used to provide data collection points within the process model so that project-specific plans that are generated using the process model are automatically instrumented to determine the success or failure of the process at those data collection points.

If a process component has failed to meet the criteria specified by the measurement model, then the user is given the option of replanning. This replanning supports rework activities by cloning user-specified portions of the activity network. The effects of rework are propagated through the project plan by creating new versions of previously created products.

Three scenarios are supported in the prototype SPMS. The newly created version of a product may be needed by tasks that have not yet started, by tasks that may be in progress, or by tasks that have already been completed. In the case of the task that has not yet begun, the new product version replaces the prior version of the product as the required input to the task. In the case where the new version is needed by an executing task, SPMS adds the new version as an input but does not remove the old product version. The individuals responsible for executing the task should be informed of the new product version. This is possible in SPMS because data on the resources and organizations to which they belong are kept in SPMS. In the case of the task that has been completed, but work needs to be performed on a new version of the product, the cloning of the activity network begins again with this task and follows the network of completed tasks within the process component. This may result in the creation of new product versions that are also propagated throughout the process model.

8.1.2.2 Project-Specific Tailoring

The process model may be tailored for use on a specific project in several ways. Elements of the coarse grained process component hierarchy may be selected for inclusion in a new process model. These selections may represent different approaches to producing the same products, such as object-oriented versus top-down design. The individual fine-grained process, product, and constraint components may also be tailored through an SQL-based interface. The product producer and product consumer activities of process components within the model may also be reviewed and edited.

The effect of the process model on the project-specific plan is also tailored by providing the development mode and architectural level parameters to SPMS when specifying those software components that are to be developed according to the process model. Those process model components that match both the architectural level and the development mode of the software components to be developed are instantiated. All other process components within the model are ignored in producing the plan.

The metrics to be computed during simulation of the plan are also selected, and the threshold values to be used as a criteria for success may be tailored. As a result of this tailoring, specific calculation methods for the expert system shell are produced.

8.1.2.3 Project Scheduling and Resource Allocation

The instantiated plan that is produced by the SPMS prototype does not include duration data for the tasks that have been created. This information is necessary before scheduling may be performed in the project management system. Resources must also be assigned to tasks before resource leveling algorithms may be applied. These user actions are done within the project management system. In the prototype SPMS this is done by exporting an ASCII file to and from the project management system.

SPMS exports information relevant to the instantiated plan, such as the node types, their relationships with other nodes, their named descriptions and, resources. The amount of information exported depends upon whether the plan has been previously scheduled, resource leveled, and simulated. If it has been simulated, it will contain resource allocations, costs, work breakdown structure, start and finish dates, scheduled dates, etc. This updates information that may already exist within the project management system. Conversely, updates within the project management system are available to be imported into SPMS.

8.1.2.4 Simulation

The scheduled project-specific plan contains all of the data that were exported to the project management system by SPMS with the scheduled start and finish dates and resources updated by the scheduling and resource allocation algorithms. This information is imported into SPMS. The plan includes scheduled start and finish dates that are based upon the task durations which were entered by the user. The durations of randomly selected tasks may be randomized by a user selected percentage. This can add a degree of realism to the simulation by letting some tasks finish early and other tasks take longer than planned to complete. During simulation, the execution of the plan may be constrained to follow the specified start and finish times or the user may allow tasks to begin as soon as their input constraints are satisfied.

The simulator may be used as a means of validating process models, the low-level metric methods that are produced to support monitoring, and the process-level metrics that may be graphically displayed during the simulation.

8.1.2.5 Adaptation - Replan or Remodel

One consequence of the simulation of the scheduled project-specific plan may be the failure of some process components to meet the validation criteria that have been specified in the measurement model. This may indicate unrealistic quality goals for some products. These goals may be tailored within SPMS. The failure may also indicate the need to perform rework on the product. SPMS supports automatic replanning by cloning user-specified portions of the project-specific plan. The cloned sections of the activity network that are required to perform the rework are automatically spliced into the activity network. Rescheduling of the plan is then likely to be necessary to assess the schedule impact of the new tasks that must be performed.

Another consequence of the execution of the simulation could be the realization that the process model is inadequate to support the proposed software development project. Problem areas within the model may be addressed by the user in an iterative cycle of model refinement, creation of new plans, and simulation of these plans until the user is satisfied that the process model will support the user's requirements.

8.2 SPMS: Port Assessment to the IBM RISC System/6000

SPMS integrates many commercial tools to provide a robust software process management capability. The following sections provide a description of these tools, their current contribution to SPMS and the issues needed to be addressed to port the current functionality to the IBM RISC System/6000.

SPMS prototype was developed and delivered by Lockheed Software Technology Center in Austin, Texas, on an Apple Macintosh system. The prototype is an integrated suite of commercial off-the-shelf tools designed to allow process definition and enactment during process project planning. SPMS combines an expert system, a relational database management system (RDBMS), hypertext interface technology and project management system capabilities (see Figure 49 on page 128). The result is a user-friendly tool that allows a user to define a process model, to instantiate the model for a specific software development project, and to simulate execution of the model on the basis of a set of software quality metrics.

The purpose of this section is to examine possible approaches for porting the prototype to the IBM RISC System/6000. The role of each tool presently in the prototype will be identified, possibilities of direct porting will be discussed, and possible alternative tools for the IBM version will be presented.

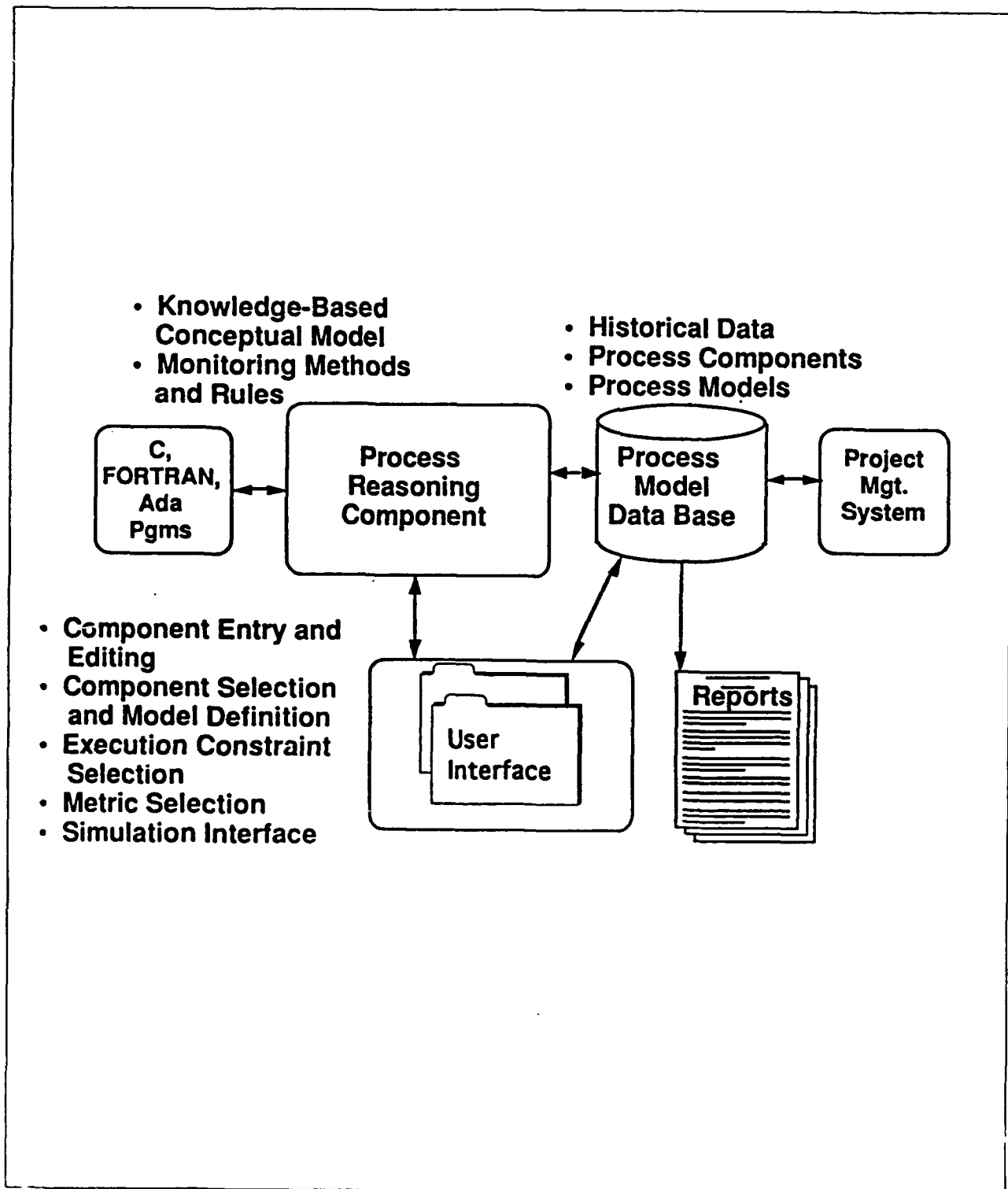


Figure 49. SPMS Architecture.

8.2.1 Process Model Database

The ORACLE relational database management system is used to provide the repository for process- and project-specific information within SPMS. The language provided by ORACLE, SQL, provides the functionality for retrieving information from the database by performing calculations on that data and dynamically creating objects during execution.

The functionality provided by ORACLE (including SQL) must be present in the ported version. RDBMS capabilities are at the core of SPMS, and any implementation would require this functionality. There are several process management database options:

1. Purchase ORACLE for the IBM RISC System/6000 and port "directly".

ORACLE is available today on the IBM platform. Cost is approximately \$16,000 to support up to 16 users. This approach seems to present the least risk we are familiar with the functionality of ORACLE and are confident that the tool provides similar functional capabilities found in the Macintosh implementation of ORACLE. Learning time will be required in order to understand the IBM RISC System/6000 version of the ORACLE products.

2. Purchase another RDBMS system and "redevelop" on the IBM RISC System/6000.

Currently Informix, Sybase, Ingres, and several other RDBMSs are available on the IBM RISC System/6000 and may provide better functionality for SPMS. We would need to investigate available products to examine their applicability for use in SPMS design and determine the cost of an alternate system. Another important factor to consider is the popularity of the RDBMS.

3. Use DBMS provided within some development/execution shell.

If IBM adopts some standard environment for use within the SEE that provides a DBMS capability, SPMS should investigate using this functionality to provide seamless integration within the SEE. Again, we would need to investigate the capabilities provided within the environment and design/develop SPMS accordingly.

8.2.2 Process Reasoning System

NEXPERT Object is an expert system shell that allows users to move from traditional data processing to knowledge processing, that permits an application to form conclusions from data and take direct action. NEXPERT Object provides a natural way of handling tasks that require problem solving or reasoning. Compared with other development tools, NEXPERT Object makes it easier to model the problem and solution space and to capture and reason with knowledge. As a result, problems can be solved that would have been very difficult with conventional languages (such as Ada and C), with the result that design, development, and maintenance are faster and easier.

Within the SPMS prototype, NEXPERT Object provides functionality in three major areas. First, NEXPERT allows an object oriented representation of processes and products. Classes (of objects) which have certain characteristics are defined. When an object of a certain class is created, it inherits all class characteristics. This includes all methods (operations) associated with the class as well as attributes and relationships to other classes. Such inheritance alleviates much programming by developers (initialization and logic) and provides much functionality to the tool with only minimal programming. Second, objects representing processes and products within SPMS must be dynamically created and deleted to model the very dynamic nature of the software development process. The memory management of these dynamic objects takes place within the expert system shell relieving developers of time-consuming programming. Finally, NEXPERT Object is used during process enactment as a shell from which to run the simulation. Rules within NEXPERT Object begin the simulation, decrease the time during task execution, start and finish tasks, calculate metrics for validation tasks, and determine whether tasks pass or fail quality checks.

As with ORACLE, the functionality provided by NEXPERT Object is central to the functionality provided by SPMS. While it may be possible to replicate the functionality provided by NEXPERT Object, it is perhaps the only readily available expert system shell that provides multiple inheritance and object-oriented development capabilities. Both of these functions contribute heavily to the ability to perform rapid systems development as well as to develop a robust IBM RISC System/6000 SPMS prototype. There are several approaches to porting this functionality to the IBM RISC System/6000.

1. Purchase NEXPERT Object for the IBM RISC System/6000 and port "directly."

NEXPERT Object is available today on the IBM RISC System/6000 platform at a cost of \$12,000.

The runtime version for the IBM RISC System/6000 is \$2,000. Its functionality is known and obviously supports all present capabilities. In the initial prototype developed by Lockheed, NEXPERT Object is not used as completely as it could be. Currently, diagnostics are not tied directly to the expert system. By exploiting the expert systems capabilities of NEXPERT object, intelligent "help" could be provided to diagnose SPMS problems. SPMS on the IBM platform should use NEXPERT Object heavily for classification knowledge modeling and process monitoring.

2. Remove expert system capabilities and replicate functionality in conventional programming language.

While this may be a possible solution, its feasibility is questionable in the current time frame. As mentioned before, expert systems provide rapid system development capabilities and alleviate much of the programming required for control logic. This choice would limit the extensibility of SPMS in terms of classification and diagnosis of process related problems.

3. Acquire required functionality from another tool within the SEE.

It may be possible to remove NEXPERT Object from SPMS and redesign it to incorporate new technology. Detailed trade studies of any potential alternatives would be necessary before making such a drastic decision.

8.2.3 User Interface

HyperCard is an application provided on Apple Macintosh systems. It is a personal toolkit with which to create applications for gathering, organizing, presenting, searching, or customizing information. It provides tools such as buttons, cards, stacks of cards, and icons to rapidly develop these applications.

SPMS used HyperCard 2.0 to develop the user interface to the tool. HyperCard provides intuitive methods for getting around within SPMS. Its point and click mechanism is easy to learn. However, HyperCard is only available on the Macintosh.

Certainly any incarnation of SPMS on the IBM platform will have to have a user interface. Just what that user interface should look like and what functionality it should provide are not so certain. There are many possibilities, several which are enumerated here.

1. Write a user interface package using the X windowing system.

While this is a possible approach, there are several already available on the IBM RISC System/6000 that would provide all functionality needed.

2. Use Motif for the user interface.

Motif is a standard windowing package provided on the IBM RISC System 6000. For developing an interface that will remain on the IBM platform, Motif is probably sufficient.

3. Develop the user interface using a product that makes the interface portable among windowing systems. XVT is a commercially available product that allows a user to build interfaces that are portable among such widely used windowing systems as Microsoft Windows, Motif, Macintosh Windows, Presentation Manager, and several character screens. XVT is available today for IBM RISC System 6000 systems at a cost of \$3,495. Interfaces built in XVT could be portable among many systems.

A product similar to XVT is UIM/X developed by Visual Edge. UIM/X is an interactive tool that allows developers to create OSF/Motif user interfaces. UIM/X's interactive development environment improves productivity in two ways:

- a. It enables developers to draw their interfaces instead of having to hand code them; and
 - b. It frees application developers from time-consuming compile, link, and debug cycles owing to its built-in C interpreter. UIM/X is not currently available on the IBM RISC System/6000, but there is a possibility that it will be made commercially available for use in an acceptable time frame.
4. Combine options 1, 2, and 3.

8.2.4 COTS Project Management System

MicroPlanner Xpert is a project management system (PMS) that provides all traditional capabilities including scheduling resources, costing, task scheduling, etc. The main reason for choosing MicroPlanner Xpert was that it readily exports data needed by SPMS. It would be preferable to have a PMS that was designed to be embedded rather than as a stand-alone tool and that utilized a relational database as its information storage rather than proprietary data structures that cannot be programatically accessed.

MicroPlanner exports, in a textual form, all information needed by SPMS to create objects that represent tasks, products, and relationships among objects. Also, SPMS can export the information back into MicroPlanner when changes are to be made to the project schedule. The integration between SPMS and the project management system is awkward and time-consuming. While the transformation of information between the two tools is automatic, it does take a great deal of time to complete. A more elegant solution will need to be designed and implemented in a final version of the tool. MicroPlanner Xpert was chosen as the project management system for the prototype, as it was the only tool readily available that exported all information needed by NEXPERT Object to create the corresponding objects.

Project management capabilities will certainly be needed in the version on the IBM platform. Complex resource and personnel scheduling algorithms are provided by any good PMS and should not have to be developed during the port to IBM equipment. We hope that a more seamless integration between SPMS and the PMS can be implemented. The approaches to be considered in the PMS area are as follows:

1. Purchase a PMS similar to MicroPlanner for IBM RISC System/6000.

MicroPlanner is not available today for the IBM RISC System/6000. We plan to examine the XPM project management system being encapsulated on HP Softbench. Given that this is not an acceptable alternative, a brief trade study of similar products would have to be undertaken to see whether a suitable alternate PMS is available.

2. Purchase another PMS that provides a callable interface or is built on SQL.

If a PMS provided a callable interface to its functionality, the user would never get the feeling of "leaving" SPMS and entering the project management system. All project management interaction could be done from within SPMS. One step further is to look for a project management system that is built on SQL so that the data used by the project management system is the same data that is created and used within SPMS. That is, all project management/process management data is stored in the same database.

3. Write our own project management system.

Although there are many commercially available systems, we may be able to write only the portions of a PMS that are needed to support SPMS. This would allow us to seamlessly integrate the PMS functions as SPMS, and the PMS would be driven by the same data (as opposed to two sets supported by stand-alone systems). The complex algorithms for scheduling may be available commercially, and simple draw packages are available. This approach would provide the most flexibility but may not be economically or technically feasible.

4. Devise a scheme for using any commercially available PMS.

If possible, the steps for including data from any PMS into SPMS could be detailed and provided to those users with in-house capabilities. Certain assumptions would be made of the PMS, such as availability of information and some type of exporting capability. SPMS would include one standard PMS and the instructions for integrating with another. This approach would make the tool attractive to organizations with their own PMS.

8.2.5 Conclusions

It may be decided that SPMS should include more functionality than the original prototype. As this is a port assessment section, those additional capabilities are only discussed briefly in the following paragraphs.

1. Exporting Process and Project Information to Other Project Monitoring Tool(s).

Tools such as EAST and KI Shell provide project guidance and monitoring for all levels of users. These tools provide process adherence, are commercially available, and usually provide many capabilities that members of a software project management/development team will need. Integration of the front-end process and project planning data from SPMS with actual process execution could provide users with a complete system for defining processes, implementing plans on those processes, monitoring the development process, and finally developing, testing and maintaining the software that they were developing.

2. Providing Coordination Technology.

One additional capability might be the inclusion of tools and/or methodologies providing the functionality for a group of users of SPMS to participate in ongoing discussions regarding the process or project. At what level within the SPMS design this might be of most importance is not clear.

3. Including Resources in Simulation and Monitoring Capability.

Currently, the process simulation provided in SPMS assumes unlimited resources (i.e., all equipment needed for a certain task is available at any given time). To truly model software development, resource information must be included in the simulation.

4. Providing Costing Information.

In the current prototype the MicroPlanner project management system exports cost information associated with each resource (personnel and equipment). This information is currently not used in the SPMS prototype but should be included so that costs can be recalculated during simulation when plans deviate from the original schedule. This information will also be useful in estimating the impact of replanning and rework.

5. Integrating with a Software Engineering Environment.

If a SEE framework is selected on which SPMS must reside which enforces certain interface or development approaches, SPMS will be integrated within this environment.

6. Integrating COCOMO Tools into the Instantiation Processing.

Integrating COCOMO tools in the SEE project management tools would provide some level of automatic generation of duration estimates, cost estimates, and resource scheduling.

7. Using Expert Knowledge and Statistical Information to Create Heuristic Rules.

Nextra, a knowledge acquisition tool from Neuron Data, will generate classification rules that can form the basis of more advanced process monitoring. This capability is currently only available on the Apple Macintosh.

8. Integrate Process Model, Metric Model, and Data Collection Model Tools into the Software Engineering Environment.

Further research is needed to understand the relationship between metrics and software development processes. Friendly interfaces are needed to add metrics pertaining to the software development process as well as consistency checking to assure that all needed information is acquired for metric calculations.

All the above approaches and alternatives are based on the final copy of the SPMS system delivered by Lockheed on June 6, 1991. We assume that the SEE is not enforcing the use of any specific commercial tools (such as a particular DBMS or project management tool). Finally, we assume that no development shell has been prescribed. If any of the above assumptions prove false, then the approaches described above may change.

8.2.6 SPMS: Port Plan for Porting SPMS to the IBM RISC System/6000

The purpose of this section is to provide an initial schedule for porting the Software Process Management System to an IBM RISC System/6000. A summary of candidate trade options is provided in Figure 50 on page 134.

		POSIX/AIX	HP SoftBench	Estimated L/M
Direct Capability Port		<ul style="list-style-type: none"> • new interface • remote MicroPlanner XPert PMS 	<ul style="list-style-type: none"> • HP SoftBench training • encapsulating SPMS 	<ul style="list-style-type: none"> • 217 I/m (Posix) • 247 I/m HP SoftBench
Re-Architect Current Product				
O P t i o n s	Replace PMS	<ul style="list-style-type: none"> • trade studies • new pms interface 	<ul style="list-style-type: none"> • POSIX/AIX issues and HP SoftBench training 	<ul style="list-style-type: none"> • 67 I/m (Posix) • 87 I/m HP SoftBench
	Write own PMS functionality	<ul style="list-style-type: none"> • trade studies to determine make or buy • new interfaces 	<ul style="list-style-type: none"> • POSIX/AIX issues and HP SoftBench training 	<ul style="list-style-type: none"> • 9-127 I/m (Posix) • 9-207 I/m HP SoftBench
	Replace Nexpert	<ul style="list-style-type: none"> • trade studies for new expert system • new interfaces 	<ul style="list-style-type: none"> • POSIX/AIX issues and HP SoftBench training 	<ul style="list-style-type: none"> • 6-247 I/m (Posix) • 6-267 I/m HP SoftBench
	Write own Nexpert Functionality	<ul style="list-style-type: none"> • trade studies to determine make or buy • new interfaces 	<ul style="list-style-type: none"> • POSIX/AIX issues and HP SoftBench training 	<ul style="list-style-type: none"> • 18-367 I/m (Posix) • 18-387 I/m HP SoftBench

Figure 50. Candidate Trade Options for SPMS Port.

8.2.7 Schedule

Figure 51 through Figure 53 describe an initial schedule for porting the SPMS capability to an IBM RISC System/6000. The task entitled "Design, Develop, Port SPMS Software to IBM RISC System/6000" is the actual porting task. The other tasks identified address developing project documentation and training materials, attending meetings, and conducting SPMS/R training sessions.

The task "Design, Develop, Port SPMS Software to IBM RISC System/6000" will incorporate many sub-tasks. Those tasks are enumerated here:

1. Trade studies on state-of-the-art commercially available tools for inclusion into SPMS.

As described earlier, SPMS combines several commercial off-the-shelf tools. Initial prototype development integrated a database management system, an expert system, and a project management system. To determine whether this suite of tools is the way we should continue must be addressed in the initial stages of the port. The following chart represents options available and issues that must be addressed when developing the final plan for the port. These options are elaborated in section 9.1 of this report.

2. Training on the IBM RISC System/6000.

There will be some learning time when the IBM RISC System/6000 is installed. As AIX is essentially UNIX, learning time should be negligible.

3. Developing new SPMS/R design.

At the very simplest level (porting directly all current technologies incorporated into SPMS), the tool will have to be redesigned somewhat when a new interface is introduced. The prototype version uses a HyperCard interface that is not available on other platforms. In the more likely case that radical redesign must take place (consider inclusion of such tools as KI Shell or HP Softbench), the redesign task will be considerably longer. Incorporating better technologies will help in the actual development, and the added time spent in design will be saved during coding. It may also be determined that additional features should be included in the design to provide a more robust, usable system.

4. Porting applicable portions to the IBM platform.

The portions that are identified as "portable" will be moved.

5. Developing SPMS/R on IBM RISC System/6000.

On the basis of technical discussions with the SPMS developers, STARS personnel, and interested third-party vendors, we believe that the correct choice for initial port is to directly port the SPMS capability as a stand-alone tool hosted on AIX. This will allow functionality in a relatively short time frame on the IBM platform. Having the tool available for beta testing quickly will provide valuable feedback for inclusion in the next version. There are still many questions that must be addressed before a robust capability that combines KI Shell, SPMS, and HP Softbench functionality can be designed and implemented. These issues are discussed in detail in section 9.1 of the Software Process Tools and Techniques Evaluation Report.

8.3 SPMS Prototype System User Training

The SEI Process Group has issued a request to use SPMS to support their software process modeling work. To gain valuable knowledge from the use of the SPMS evaluation prototype, IBM provided SEI with a training class in the use of the SPMS prototype. Support of the use of the SPMS evaluation prototype is planned, at a limited level, through the STARS "T" increment.

A training course has been designed to instruct users of SPMS. The course is set up to run for one-and-one-half days and gives students opportunity to build a process model, tailor the model, create a plan from the model, and finally simulate the plan. The following section describes course content and handouts.

8.3.1 SPMS Training Materials and Discussion

On the first day of the training class, students are introduced to SPMS, given background on the terminology to be used, a discussion of process models and project plans, and introduced to building a model by using the graphical representation facility. Next, the user will import this model into SPMS and tailor it according to his or her personal preference. Finally, the user will create a project plan based on his or her modified model, schedule the plan, and simulate plan execution.

On the second day of class, students will validate tasks in the plan, and on the basis of the outcome of validation may rework the model or project plan. More complex models will be introduced, and the previous steps repeated. The class will close with discussion and questions. The class will perform six exercises on SPMS.

The class will be given a "generic" process model to input. This example is based on a draft ISO standard for software life cycle processes. The hardware and software needed for the class are identified in the following section.

8.3.2 SPMS Evaluation Prototype: Hardware/Software Requirements

The purpose of this section is to provide the hardware and software requirements for installing the SPMS system evaluation prototype. An understanding of how to use a Macintosh computer is assumed as well as a knowledge of the components of the SPMS system. These components are the ORACLE RDMBS, NEXPERT Object expert system shell, HyperCard, and MicroPlanner Xpert.

8.3.2.1 Hardware Requirements

All of the components of this system are capable of running on most Macintosh computer systems (from the Macintosh Plus to the Macintosh IIfx). The only limiting factors affecting the decision upon which computer system to select for the installation of the SPMS software are the amount of memory in the computer system, the amount of hard disk space that is available to the computer, and the performance capability of the computer selected.

The SPMS system needs approximately 9 megabytes of RAM to execute the plan provided with the system (Plan-1). This plan contains 75 products, 240 tasks or milestones, and 448 constraint relations. A larger plan would need more memory. The hard disk on which the SPMS system is placed will need 6 megabytes of storage for the software to be installed. ORACLE's disk partition should be expanded an additional 10 megabytes (20,480 partition units) - not including the 5 megabytes needed for ORACLE itself, which brings the total up to 15 megabytes of storage for ORACLE. The 10 megabytes of expansion will provide sufficient room to load and work with the database tables and build some new models and plans. Finally, plan for an additional megabyte of storage for NEXPERT's NDL interface. This provides a total of 22 megabytes of hard disk storage for the SPMS system (6 for SPMS, 15 for ORACLE, 1 for NEXPERT).

The final factor to consider when deciding what type of Macintosh computer system on which to install the SPMS system is the system performance required. SPMS was developed on Macintosh IIfx computers, with 160MB of hard disk, 20 megabytes of RAM, and two monitors - a 21" two-page monochrome and a 13" RGB monitor. Theoretically, each of the components of the SPMS system (HyperCard, NEXPERT, ORACLE, and Microplanner) will run on a Macintosh Plus; therefore, providing the requirements stated previously are met, SPMS should run on a Macintosh Plus. Performance requirements must be addressed when considering that the SPMS system is reasonable when executing on a Macintosh IIfx. It is recommended that a Macintosh II class computer system (Macintosh II, IIfx, IIsi, IIfx, IIsi, IIfx) be selected, to be

accompanied by a large monitor (19" or larger or multiple monitors). Use of a smaller screen will result in poor display performance and display jittering, which makes small display use cumbersome and inefficient.

8.3.2.2 Software Requirements

The SPMS system requires ORACLE for the Macintosh v1.2, NEXPERT v2.0B, and Microplanner Xpert v1.0.4. Development started with system v6.0.3 and finished with System v7.0. The only system version requirement comes from the dependencies of the components of the SPMS system (HyperCard, etc.).

8.3.2.3 Installation of the SPMS System

The SPMS Evaluation Prototype system is distributed on 7 diskettes. They are SPMS-1, SPMS-2, SPMS-3, SPMS-4, SPMS-5, SPMS-6, and NDLCClient "Execute" Stack. This software can be installed anywhere on the computer system's hard disk, as long as the hierarchical structure illustrated in Figure 54 is observed.

SPMS				
8 items		130.8 MB in disk		23.3 MB available
Name	Size	Kind	Last Modified	
▶ Execute Plan	403K	folder	Thu, May 2, 1991, 3:24 AM	
▶ Headers	23K	folder	Sat, Apr 20, 1991, 3:10 AM	
▶ Models	773K	folder	Sun, Apr 28, 1991, 2:31 AM	
SPMS stack	805K	HyperCard document	Wed, May 1, 1991, 5:23 AM	
▶ Stacks	815K	folder	Sat, Apr 27, 1991, 4:55 AM	
▼ Xpert	1,283K	folder	Thu, May 2, 1991, 2:51 AM	
▶ Exports from SPMS	103K	folder	Wed, Apr 24, 1991, 9:57 PM	
▶ Exports from XPERT	1,180K	folder	Sun, Apr 28, 1991, 1:33 AM	

Figure 54. Hierarchical Structure of SPMS Folder.

To begin, create a folder on the computer system's hard disk and provide it an acceptable name. In Figure 54, it is named "SPMS" (the name of the window). The SPMS system does not depend upon the name of this folder, but it does depend upon the naming and placement of the folders within the SPMS folder. Disks SPMS-1 and SPMS-2 contain files that are exports from ORACLE. These files, named spms.dmp.1 through spms.dmp.9, need to be imported into ORACLE by using the Import card for the System Stack that was provided with ORACLE. Before the export files are imported, replace the init.ora in

the Drivers folder of the ORACLE home folder with the init.ora provided on disk SPMS-2. This file adjusts some of the default system variables so that SPMS's tables will load into and run under ORACLE.

Before executing the SPMS system, a folder must be allocated for SPMS.

The location of the files shown in Figure 54 are as follows:

- The SPMS stack and Headers folder are located on disk SPMS-2.
- Disk SPMS-3 contains the Execute Plan and Stacks folder.
- The folder Models resides on disk SPMS-4.
- Because of the size of the folders contained within the Xpert folder - Exports from SPMS and Exports from XPERT - the Xpert folder is divided between disk SPMS-4 and SPMS-5.

A folder must be created within the SPMS home folder with the name *Xpert*. All exports from the SPMS folder will be placed in the Xpert folder. The exports from the SPMS folder are contained in the folder Xpert.part.1 on disk SPMS-4. Also, the folder Exports from XPERT needs to be placed with the Xpert folder. It is located on disk SPMS-5 in the folder Xpert.part.2. The last two disks contain additional stacks that are optional to install but are not needed to run the SPMS system. The disk SPMS-6 contains stacks that permit access to the metrics functionality of the SPMS system directly. Briefly, the stacks on this disk are the

- Generator stack, which allows the generation of the ORACLE tables used to hold metrics data and for entering the structure of the project upon which the metric measurements will be performed;
- Software quality stack, which gives access to the questions used to collect data for the metrics; and
- Calculator stack, which will allow the calculation of selected metrics.

The disk NDLCClient "Execute" stack contains an alternative version of the stack used to execute a plan. This version uses a client-server version of the NEXPERT system that uses the NEXPERT Development environment instead of the NEXPERT Runtime environment for its processing. To use this stack, replace the stack contained with the Execute Plan folder in the one on this disk.

At the time of this writing, a bug in the NEXPERT runtime prevented the use of the Execute stack so that the only way to gain access to the functionality of the Execute stack was to use the NDLCClient stack on this disk.

8.3.2.4 Summary

In summary, the installation of the SPMS system requires the following items:

Hardware:

- 9 Megabytes of RAM for the SPMS system to run (required for NEXPERT Development Environment)
- 6 Megabytes of RAM for the SPMS system to run (required for NEXPERT Runtime) when its bug is fixed
- 22 Megabytes of hard disk to install SPMS (required)
- 19" or larger monitor (recommended)
- Macintosh II class computer system (recommended)

Software::

- ORACLE for the Macintosh v1.2 (required)
- NEXPERT v2.0B with ORACLE authorization (required)
- Xpert v1.0.4 (required)
- HyperCard 2.0 (required)
- System v7.0 (required for use with NEXPERT Development Environment).

Also, the software provided on the disks SPMS-1 through SPMS-5 needs to be installed within a folder using the hierarchical layout shown in Figure 54 on page 140.

8.4 Major Lessons Learned from SPMS Migration Analysis and SPMS Training

This section summarizes major lessons learned from the SPMS Migration Analysis work and from preparing for and teaching a class in the use of the SPMS evaluation prototype.

8.4.1 Lessons Learned from SPMS Migration Analysis

From our analysis of what is required to port the SPMS prototype system from the Apple Macintosh to an IBM RISC System/6000 we have learned several lessons that need to be conveyed to CASE vendors and framework developers:

1. Project management systems can no longer be viewed as "stand-alone" systems that only project management and planning teams' use. They must be provided with programmatic interfaces that permit their effective integration with tools that support process management and software development. Further, project management system database standards should be developed to permit standard open access to project management data that other applications can access and update.
2. Control integration tools such as HyperCard permitted SPMS to be developed in an accelerated timeframe, as the SPMS developers were able to take advantage of products that provided programmatic interfaces for HyperCard integration. Selection of a suitable control integration mechanism for the RISC System/6000 and lobbying vendors to provide programmatic interfaces to facilitate tool integration are essential. The success of HyperCard should be used as a model by SEE Framework vendors and CASE application providers.

9.0 IBM STARS SEE Process Management Architecture Discussion

Exactly what will be included in a final version of a process management system is not completely clear at this time. Several strategies have been discussed and reviewed. The following sections describe possible integration strategies, including issues to be addressed and a candidate IBM STARS process management architecture.

9.1 SPMS Coexistence Strategy with Other Process Management Capabilities

The SPMS prototype has been described in detail in previous sections of this report (8.1.1, 8.2). For future process design/modeling/enactment capabilities other process management facilities may be required to be integrated with SPMS. The purpose of this section is to describe two systems that may provide process control capabilities, integration facilities and support for groupware functions needed to provide a robust process management capability. We describe Hewlett Packard's (HP) SoftBench and UES's KI Shell and present issues regarding a coexistence strategy for all three tools.

9.1.1 HP SoftBench

HP SoftBench 1.0 is a software development environment consisting of both an integrated set of program development tools and a Tool Integration Platform. HP SoftBench provides five tools that target the program construction, test, and maintenance phases of software development. In addition to these tools, which are standard in SoftBench, users can use the Tool Integration Platform, which allows for integrating other tools into the environment. The Tool Integration Platform is responsible for providing distributed computing services, tool communication, OSF/Motif appearance and behavior across all tools, and integrated on-line help facility.

Distributed Computing Services

The HP SoftBench tools can execute in any host in the network, provided that the host has HP SoftBench installed. The remote execution is transparent to the client tool that requested its services. Distributed execution provides full utilization of network resources. With HP SoftBench, data can reside on any host in the network. A tool is able to access the desired data independent of where the tool is running. To implement this transparent data access, HP SoftBench uses either Network File System (NFS) or IIP's Remote File Access (RFA), depending on which is available on the system. IIP SoftBench is built on the X Window System, version 11, an industry standard. This allows programs to execute on one system and display and support user I/O on another.

Communication

The IIP SoftBench tools communicate in a networked, heterogeneous environment via a broadcast communication facility designed to support close communication of independent tools. Message requests allow one tool to invoke the functionality of another tool, and notification messages allow tools (or the user) to define triggers that respond to events and initiate other actions. Tool communication allows users to customize and extend the IIP SoftBench environment.

Users may bring their own tools into the IIP SoftBench programming environment by using the IIP Encapsulator. One or more tools may be linked together to support a task or process. The results of the encapsulation process are an encapsulated tool with a consistent user interface based on the OSF/Motif appearance and behavior and the ability for that tool to communicate with the other IIP SoftBench tools.

OSF Appearance and Behavior across All Tools

HP SoftBench provides a multiwindow graphical user interface. This allows for an easy-to-learn system that requires little need for reference to documentation. Productivity is increased by having a consistent user interface across all tools. HP SoftBench implements the OSF/Motif appearance and behavior adopted by OSF as an industry standard.

Integrated Help Facility

The help facility cooperates with the other tools in the environment to service the user's request for help. Help can be obtained for general information, context-sensitive information, and definitions of terms used by any of the HP SoftBench tools.

9.1.2 KI Shell

KI Shell is an object-oriented environment for creating and executing software assistants that provide "decision support." The KI Shell development tools are used to represent (in a declarative, machine-readable form) a method by which any end-user can progress through a complex collection of activities and use multiple, existing software applications. An example of this is a prescribed process that must be followed by several persons on a project.

The runtime KI Shell Library is a collection of generic utilities designed to interpret a method representation created by the KI Shell development environment.

The term assistant is used to describe the final, KI Shell-generated software, which comprises of a method representation, interfaces to applications, and the runtime KI Shell utilities. The assistant aids the end-user in the completion of activities according to the represented method (or process).

KI Shell is used by those persons who specify the method (process), a programmer who implements the method via C language programming, and the final user who must follow the method (process) prescribed.

9.1.3 SPMS, KI Shell, and HP Softbench Coexistence Strategy

The coexistence of SPMS, KI Shell, and HP Softbench will require resolution of many issues. Through discussions among SPMS designers and implementers, KI Shell developers, STARS personnel as well as reading HP Softbench literature and attending demonstrations, we recognized and raised the following issues:

1. KI Shell provides enactment capabilities for executing processes. SPMS provides simulation of executing processes. The ability to integrate both tools is desirable.
2. KI Shell provides a role-based view of a process, while the SPMS provides an activities based view of the process. Both are necessary, however neither is sufficient by itself. A tool combining both (and perhaps other) views is desirable.
3. Both tools currently use the same database management system (DBMS). There is very likely some overlap in the data being kept by each tool as well as differences in the data representations used by each.
4. SPMS provides some automatic simulation of a process as process development continues. KI Shell relies on hand simulation of a process to validate the steps. It would be desirable to integrate SPMS automation techniques with KI Shell enactment capabilities.
5. KI Shell provides a rich set of control structures for navigating through a process. SPMS allows conditional replanning by instantiating portions of the process.
6. HP Softbench provides inter-tool communication capabilities. Both tools could take advantage of this functionality.

7. KI Shell makes decisions and coordinates activities based on low level activities such as availability of inputs. SPMS provides more global information and coordination based on not only availability of inputs but also schedule and resource allocation. Both capabilities are needed in a final system.
8. KI Shell develops a process model via a frames editor that has knowledge control structures, data storage, and different roles. The SPMS uses the front end of XPert project management system to lay out tasks and products as well as their relation to each other. How these two representations can be combined is an issue to be resolved.
9. KI Shell uses system-level calls from within C routines to execute applications. SPMS includes an expert system shell which could be used to execute applications. HP Softbench will be Portable Common Tool Environment (PCTE) Tool Interface Standards compliant. Both SPMS and KI Shell could make calls to Softbench to become portable among systems.
10. Neither SPMS nor KI Shell provides a process architecture classification standard. If a classification scheme is derived, should both tools be compliant?

9.2 IBM STARS Process Management Architecture Options

There are several architecture options that could be selected for providing a software process management capability for the IBM STARS SEE. These options include:

1. Stand-alone SPMS and KI Shell under AIX

This option involves the non-integrated operation of both SPMS and KI Shell executing on top of the UNIX operating system. SPMS would support activity-based process modeling. KI Shell would support role-based process modeling and the implementation of a system to invoke tools, based on the process requirements for the tasks allocated to each user role.

2. Loose integration between SPMS and the KI Shell under AIX

SPMS would support activity-based process modeling. KI Shell would support role-based process modeling and the implementation of a system to invoke tools, based on the process requirements for the tasks allocated to each user role. In addition, KI Shell would maintain process state information, process metrics, and process activity information that could be used by SPMS to monitor and track the process being supported by KI Shell against the activity-based process model for a project. Integration would be accomplished by sharing an ORACLE-based process state and history database.

3. Integration of both KI Shell and SPMS on top of a software engineering environment, such as HP SoftBench.

In addition to what is provided by the loose integration of KI Shell, integration into a software engineering environment framework provides the ability to more tightly integrate SPMS, KI Shell, and an appropriate project management tool. The process integration capabilities provided by KI Shell are beyond the capabilities of most currently planned software engineering environment frameworks. The System Integration Library of KI Shell could be reimplemented to use the software engineering environment's services, and would provide a more robust process management capability than that of most traditional integration services. Further, making KI Shell PCTE-compliant and migrating it to a PCTE-compatible software engineering environment framework would ensure that KI Shell process system applications could potentially be rehosted on any UNIX workstation for which there is a PCTE-compliant software engineering environment framework.

In the case of HP SoftBench, the KI Shell Systems Integration Library (SIL) could be reimplemented to employ HP SoftBench control integration services after HP SoftBench has been upgraded to support the ECMA PCTE Tool Interface Standards. A KI Shell application is developed by writing a C host program that employs the SIL to develop process systems. An HP SoftBench application encapsulation can be accomplished in a similar manner, employing the "encapsulator's" control integration services. By making the KI Shell services consistent with those provided by HP SoftBench's service, more robust

process systems can be developed, that not only integrate applications in a software engineering environment and automate execution steps, but automate the processes governing their use.

The use of a framework which provides data integration services, is not aggressively being pursued at this time. However, if a framework is selected that provides us with a PCTE data integration capability, we will assess this capability and revise our process support environment plans, as required.

The selection of these options is dependent on the software engineering environment framework selected by the IBM STARS SEE task.

9.3 Product Integration Strategy

Like most process related tools, both KI Shell and SPMS must provide modeling features. SPMS uses MicroPlanner for this purpose, while KI Shell also provides its own interface for process modeling. SPMS has features for process simulation. KI Shell has features for process enactment and process management. In this section, we will present an integration strategy.

9.3.1 The Components for a Process Support Environment

In this section, we will discuss the components required for a Process Support Environment (PSE).

Project Planning Component: A Process Support Environment should provide services to support the planning of software development efforts. Our candidate tool to support software project planning is MicroPlanner. Process planners and process engineers will work to identify and develop project activities based on the project objectives to be accomplished and the processes that will guide how project activities should be performed.

Activity-Based Process Modeling Component: A Process Support Environment should provide services to support the modeling of software processes. Our candidate tools to support activity-based process modeling are SPMS and AMS¹². The project plan prepared in MicroPlanner will be imported into SPMS. Process engineers will take the project plan and use SPMS to instantiate it, based on existing or planned processes. AMS will be used to assist process engineers identify reusable process assets to support their process modeling tasks.

Process Model Simulation Component: A Process Support Environment should provide services to support the testing of software processes before they are installed for use. Our candidate tool to support process model simulation is SPMS. SPMS provides process engineers with this capability.

Process/Project Planning Support Component: A Process Support Environment should provide services to support the integration of existing process and project data to facilitate process model and project plan refinement. Our candidate tools to support process and project planning support are SPMS and MicroPlanner. SPMS provides process engineers and project planners with the ability to incorporate the data necessary to prepare project cost estimation models, such as COCOMO. Further, data available to SPMS may be used to update the project planning data in the project planning and management tool, e.g., MicroPlanner.

Role-Based Process Modeling Component: A Process Support Environment should provide services to support the development of role-based process models. Our candidate tool to support role-based process modeling is the KI Shell development environment. Further, it may be helpful to augment KI Shell's devel-

¹² The IBM STARS Asset Management System (AMS) is being developed by the IBM Reuse Team composed of members from IBM FSD and SAIC.

opment capability with a suitable CASE tool to support systems analysis and design. The KI Shell development environment provides process engineers with tools to allocate activity-based processes to project roles. The role-based process model will provide knowledge of the project activities that have been assigned to all project role categories, such as process engineer, project manager and software developer.

Process System Development Tool Component: A Process Support Environment should provide services to support the implementation of role-based process models. Our candidate tool to support process system development, e.g., implementation of the role-based process models as an executable system, is the KI Shell development environment. Further, depending on the host implementation language selected, integration function libraries, such as ORACLE's PRO*C may be required. The KI Shell development environment provides process engineers with tools to implement the specified process models as an executable process system.

Process Enactment Component: A Process Support Environment should provide services to support the enactment of the project's process. The KI Shell-based process system developed will serve as the mechanism for ensuring process, tool and application system availability to all project members. SPMS may be called on as a service from a KI Shell process system to support given process steps, such as computing and providing product metrics.

Process Monitoring Component: A Process Support Environment should provide services to support the monitoring of the project's processes, in order to provide support for measurement collection, and ultimately process improvement. The KI Shell application will provide "tactical" process enactment support and record events in the "process state and history database." SPMS will provide "strategic" process management support by monitoring the software process for aggregate trends requiring process analysis and improvement.

An overview of the process components identified above is provided in Table 3.

Components	Candidate Tools
Project Planning	MicroPlanner
<ul style="list-style-type: none"> o Plan project activities. o Define resources. o Develop schedule. o Define activity exit criteria. 	
Activity-Based Process Modeling	MicroPlanner, SPMS, AMS
<ul style="list-style-type: none"> o Develop activity-based process models. o Incorporate reusable process components in activity planning. o Refine resources. o Refine schedule. o Refine activity exit criteria. 	
Process Model Simulation	SPMS
<ul style="list-style-type: none"> o Examine consumption of resources. o Ability to view different execution paths prior to enactment. 	
Process/Project Planning Support	MicroPlanner, SPMS
<ul style="list-style-type: none"> o Refine activity-based process models. o Provide project and process decision support. o Support process data integration for cost model generation. o Refine resources. o Refine/revise schedules. 	
Role-Based Process Modeling	KI Shell
<ul style="list-style-type: none"> o Identify and model project personnel role types. o Allocate project activities to role types. o Identify resource requirements to satisfy each allocated activity. o Analyze how each activity could be meaningfully measured. 	
Process System Development Tool	KI Shell, SPMS
<ul style="list-style-type: none"> o Design a system to support the role-based processes identified. o Identify requirements for tool integration, in either KI Shell or the selected SEE framework. o Implement the processes as a KI Shell process system application. 	
Process Enactment Component	KI Shell, SPMS
<ul style="list-style-type: none"> o Support process-driven software development. o Provide data for decision support for activity execution. o Enforce process enactment discipline between roles. o Accumulate actual process execution data. o View and report on process enactment status. o Maintain the "process state and history database." 	
Process Monitoring Component	SPMS, KI Shell
<ul style="list-style-type: none"> o Support "strategic" process monitoring. o Watch for aggregate process trends. o Support process project plan update, based on task results. o Generate reports for process improvement analysis. 	

Table 3. Components of a Process Support Environment

9.3.2 Process Support Environment Integration

Of the options identified, IBM will likely integrate the identified process components either as independent, loosely integrated applications running on top of AIX or integrated into a SEE framework. Both options will be briefly discussed.

9.3.2.1 A Process Support Environment Integrated on Top of AIX

The tools of SPMS, KI Shell, ORACLE, AMS and MicroPlanner can be loosely integrated to provide a process support environment. A loose integration strategy is an appropriate candidate, while SPMS is still in the state of product prototype development. Control integration of applications could be handled through the use of UNIX shell scripts, specially developed product bridges, or through limited use of KI Shell's product integration facilities.

Data integration could be handled through the use of ORACLE as a persistent "process state and history" repository. SPMS will be developed based on the integration of NEXPERT Object and ORACLE, and already has a facility to import data from MicroPlanner. KI Shell is a COTS tool, which uses ORACLE as a persistent database. Using ORACLE as a common persistent database would facilitate data integration between KI Shell and SPMS. MicroPlanner is already set up to import and export table data. Further, NEXPERT Object provides an ORACLE bridge. As a result, investment in data integration bridges should be minimized.

Alternatively, standard application programming interfaces could be provided by the KI Shell to permit SPMS to create and store appropriate process data in the process repository. KI Shell could also provide the interface to invoke the SPMS tool within the "process management" role of any KI Shell method. Other roles of a KI Shell method could be designed to write critical execution metrics in the process repository. The "process management" role could be developed to employ this data, as well as make it available to SPMS, to support its process management role of process monitor.

A KI Shell application could be developed to provide process integration for a project, including the process engineering and project management functions, based on the processes they perform and the tools they require.

9.3.2.2 A Process Support Environment Integrated into HP SoftBench

Control integration between a KI Shell process application and the selected Process Support Environment tools could be accomplished through the use of HP SoftBench's message service. Figure 55 on page 150 provides an architectural view of the Process Support Environment's components, communicating through the message service, while data integration is accomplished either through ORACLE or on a tool-to-tool basis, as described in the previous section. Please note that Figure 55 on page 150 takes a product view and that the Process Support Environment components identified earlier are functionally integrated into the architectural components shown.

Process integration, as identified above, to support project use, could be managed through a KI Shell process application.

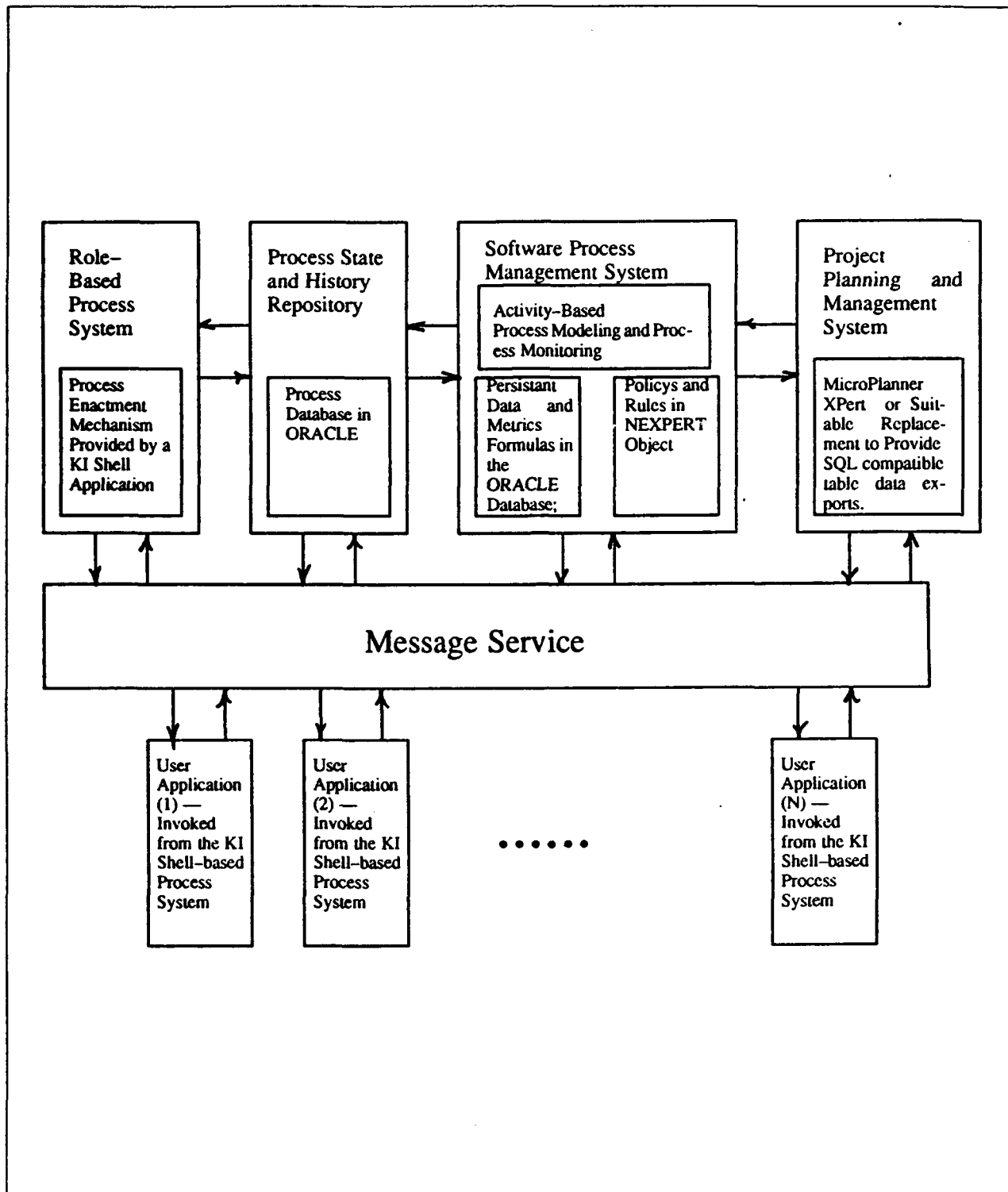


Figure 55. Candidate IBM Process Support Environment Architecture Concept.

The "role-based process system" block shown on Figure 55, represents a KI Shell application to support the PSE process enactment component. The "process state and history repository" block shown on Figure 55, represents the persistent store of process data that is to be shared by the KI Shell process application and SPMS. The "Software Process Management System" and "Project Planning Management System" blocks shown on Figure 55 satisfy the following PSE components:

- Project Planning

- Activity-Based Process Modeling
- Process Model Simulation
- Process/Project Planning Support
- Process Monitoring.

The PSE components not shown on Figure 55 on page 150 are the "Role-Based Process Modeling Component" and the "Process System Development Tool Component." These components are satisfied in part by the KI Shell development environment, and suitable COTS CASE tools, such as CADRE TEAMWORK and Statemate. Both activity-based and role-based process modeling may be facilitated by using AMS to help identify candidate process assets for reuse.

9.3.3 Benefits of a Process Support Environment

Several advantages result from a tool set integrated into a process support environment. With data integration -- all process tools work on the same database -- it is possible to use the same current data. This results in many advantages:

1. Simulation can use actual project execution data preserved by the enactment tool. Therefore, simulation provides more accurate project status for process management and for replanning. This facilitates continuous process improvement.
2. Process management is based on actual resource usage, because consistent data is used by all tools. This avoids cost overruns.
3. There is global awareness of exact process status. This allows early resources to be redirected to problem areas.
4. By enacting a modeled process and capturing precise model-based metrics in the database, real process improvements are easier to identify.

KI Shell's Process State Data contains a role, activity structure, and attribute data. Program interfaces can be provided to all tools that are to be tightly integrated. The integrated process support environment depends on three levels of integration, namely:

1. **Data Integration:**

Common, consistent, up-to-date data available for all tools.

2. **Control Integration:**

Standard way to invoke, suspend, and pass data to a tool.

3. **Process Integration:**

Discipline by which users collaboratively enact steps.

These three levels of integration are illustrated in Figure 56 on page 152.

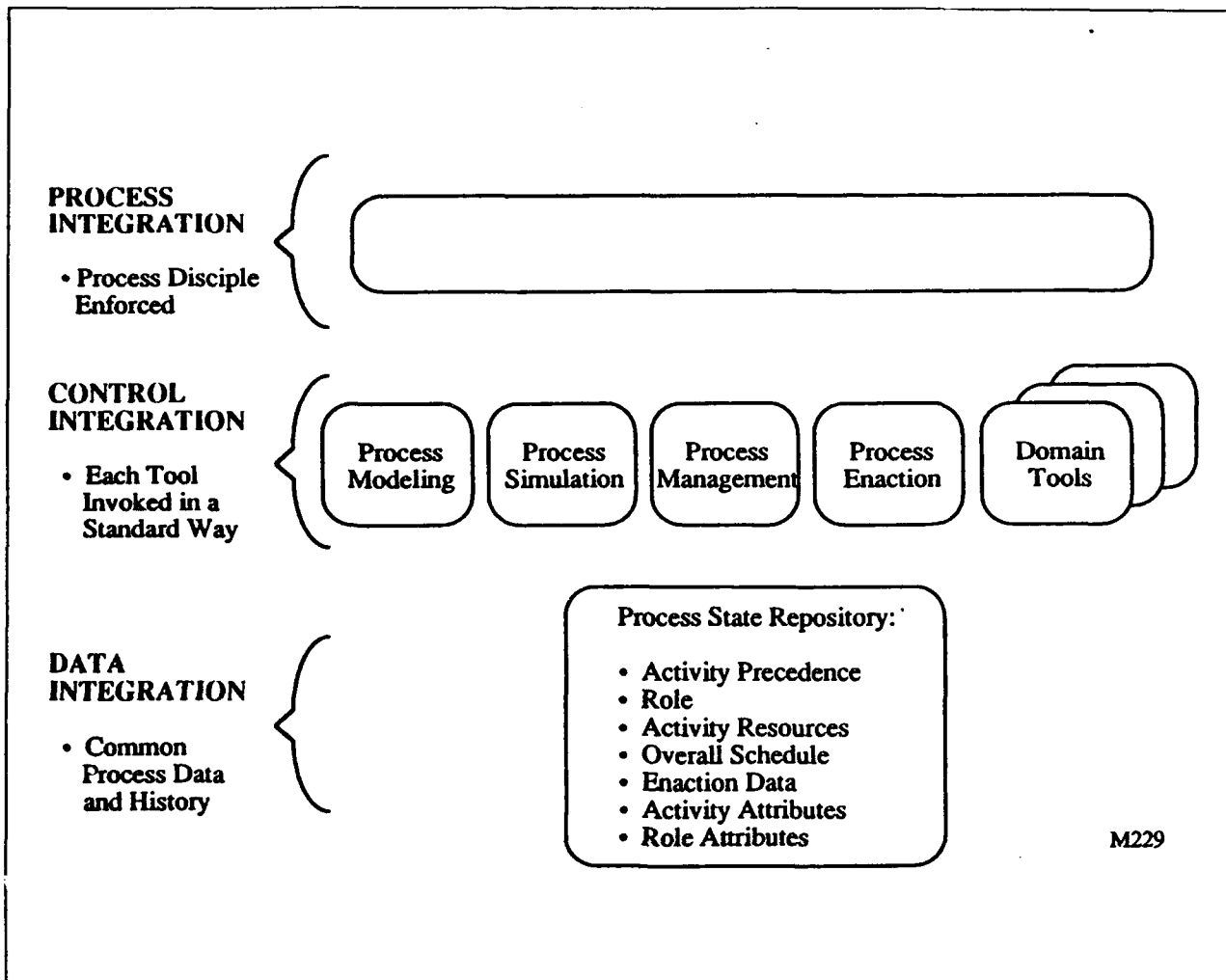


Figure 56. Levels of Integration.

10.0 IS-15 Task Conclusions

From conducting the "*Cleanroom Software Process Case Study*" we reached the following conclusions:

1. **A pre-condition for implementing a process for support by a process enactment mechanism is a well-defined process.**

The "*Cleanroom Engineering Software Development Process (SDP)*" is a very well-defined process. The Cleanroom Engineering SDP was prepared for humans to follow in performing Cleanroom Engineering to develop software systems. However, the Cleanroom Engineering SDP needed to be expressed in a sufficient level of detail to enable its programming as a KI Shell application, namely CEPA, to support the enactment of the Cleanroom Engineering SDP.

2. **It is possible to implement complex processes in the KI Shell.**

The "*Cleanroom Engineering Software Development Process*" was a very suitable example for the KI Shell, in that it was able to take advantage of the its representation and processing power. KI Shell was developed to accommodate sophisticated processes where:

- a. Each process is composed of multiple steps
- b. There is synchronization of processes among process steps
- c. There is cooperation among processes steps (e.g. one step may produce a result usable by another process step, etc.)
- d. Process steps require the invocation of utility or application programs
- e. Process steps have a reasonably complex control structure.

3. **It is possible to implement a complex KI Shell process application in a reasonable time frame.**

The final version of the CEPA prototype system, based on the CEPA specification, was implemented in 1 man month. Much of the time spent on the case study problem was in understanding the problem, not implementing it.

4. **The use of Box Structure notation for recording processes as exemplified in the "*Cleanroom Engineering Software Development Process*" is excellent for conveying process requirements.**

The case study implementation team was able to read this process document and interpret the process notation without formal training. However, it was expressed that "box structures reader training" would have been helpful. However, Box Structure notation is not all that is required to plan processes for development. Role specification and planning techniques are also extremely important.

From conducting the "SPMS/KI Shell Coexistence Study," we reached the following conclusions¹³ :

1. **Development of process models for a software development project require "activity-based" process modeling as well as "role-based" process modeling.**

The Software Process Management System provides the ability to combine the process model for a project with project planning data to produce the "*Project/Process Plan*." The KI Shell allocates project

¹³ It should be recognized that the "SPMS KI Shell Coexistence Study" is based on our current understanding of the state of both the SPMS product and KI Shell, and the identified COIS tools. Further, the coexistence strategy suggested, may have to be revisited several times before a final "Process Support Environment Solution" architecture is realized.

activities to specific roles assigned the responsibilities to perform project activities and support people who assume these roles in following the prescribed process.

After the network of activities for a project have been identified, they can easily be allocated to roles, and methods for user role support of process activities (or process tasks) can be developed.

2. Processes, at the "activity level" can be simulated before deploying them for use.

The Software Process Management System provides the ability to simulate a process to determine how it will perform under specified conditions. The results of the simulations can be used to correct or improve a process model, before it is implemented for enactment.

3. SPMS and the KI Shell can provide complementary support in the preparation of a STARS process management solution.

The Software Process Management System can provide support for process activity modeling. KI Shell can provide support for role modeling and process enactment. SPMS can provide a simulation and monitoring capability. KI Shell can provide tactical control over the execution of the process application developed, and could make available process and product state data, and event data to SPMS to permit the tracking of the planned "process/project plan" versus actual project performance. Where the KI Shell provides users with knowledge of what tasks to perform and provides process guidance to perform the tasks, the SPMS can provide the users knowledge of when to perform their tasks. We feel that the combination of these two powerful capabilities will provide an excellent basis for fielding a software process management capability.

4. The product and process metrics available in SPMS and KI Shell will help facilitate a process measurement and improvement capability for management.

KI Shell provides a built-in capability to collect process metrics within process steps and report them for analysis. Further, it is possible to build in to KI Shell process applications, the execution of tools to provide product metrics for collection and reporting. SPMS has integrated the RADC Quality Framework product measurements and makes them available for use in developing activity-based process models. The combination of these capabilities provides process engineers with a set of tools for collecting and analyzing measurements collected from performing process steps and measurements collected about the products of process steps, to support a project's process improvement activities.

11.0 References

1. Adams, E. N., "Optimizing Preventive Service of Software Products," IBM Journal of Research and Development, January 1984.
2. Boehm, B. W., et al., "Characteristics of Software Quality," North Holland Publishing Company, 1978.
3. Boehm, B. W., "Software Engineering Economics," Prentice-Hall, 1981.
4. Boehm, B. W., "Improving Software Productivity," IEEE Computer, Volume 20, Number 9, September 1987, pp. 43-57.
5. Boehm, B. W., "The Spiral Model of Software Development and Enhancement," IEEE Computer, Volume 21, Number 5, May 1988, pp. 61-72.
6. Brooks, F. P., "No Silver Bullet: Essence and Accidents of Software Engineering," IEEE Computer, Volume 20, Number 4, April 1987.
7. Cobb, R. H. and H. D. Mills, "Engineering Software under Statistical Quality Control," IEEE Software, November 1990.
8. Curtis, B., H. Krasner, and N. Iscoe, "A Field Study of Large Software Projects," Communications of the ACM, Volume 31, Number 11, November 1988.
9. Cusumano, M., "Hitachi: Pioneering the Factory Model for Large-Scale Software Development," MIT Sloan School, Working Paper 1886-87, Cambridge, Massachusetts, 1987.
10. DeMarco and T., T. Lister, *Peopleware*, New York: Dorset, 1987.
11. Dyer, M. and A. Kouchakdjian, "Correctness Verification: Alternative to Structural Software Testing," Information and Software Technology, January/February 1990, pp. 53-59.
12. Green, S.E., A. Kouchakdjian, and V. R. Basili, "The Cleanroom Case Study in the Software Engineering Laboratory: An Experiment in Formal Methods," SEL, University of Maryland, 1989.
13. Humphrey, W. S., "Characterizing the Software Process: A Maturity Framework," IEEE Software, March 1988, pp. 73-79.
14. Kellner, M. I., and H. D. Rombach, "Session Summary: Comparisons of Software Process Descriptions," Proceedings of the 6th International Software Process Workshop: Support for the Software Process, Hakodate, Japan, ACM Press, October 29-31, 1990.
15. Kling, "The Web of Computing: Computer Technology as Social Organization," Advances in Computers, Volume 21, pp. 1-90, Reading, Massachusetts: Addison Wesley, 1982.
16. Krasner, H., B. Curtis, and N. Iscoe, "Communications Breakdowns and Boundary Spanning Activities on Large Programming Projects," in Proceedings of the Second Workshop on Empirical Studies of Programmers, pp. 47-64, Norwood, New Jersey: Ablex Publishing, 1987.
17. Lehman, M. M., and L. A. Belady, eds., "Program Evolution: Processes of Software Change," APIC Studies in Data Processing, Volume 27, London: Academic Press, 1985.
18. Linger, R. C., H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice*, Reading, Massachusetts: Addison Wesley, 1979.
19. Linger, R.C. and H.D. Mills, "A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility," Proceedings of COMPSAC '88, IEEE, 1988.
20. Linger, R. C. and H. D. Mills, "A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility," Proceedings of COMPSAC '88, IEEE, 1988.
21. McCall, J., "Factors in Software Quality," GE-TIS-77CIS02, General Electric Company, 1977.

22. McGarry, F., "What Have We Learned in the Last Six Years?" Proceedings of the Seventh Annual Software Engineering Workshop (SEL-82-007) Greenbelt, Maryland: NASA GSFC, 1982.
23. Mills, H. D., "Stepwise Refinement and Verification in Box-Structured Systems," IEEE Computer, Volume 12, Number 6, June 1988.
24. Mills, H. D., R. C. Linger, and A. R. Hevner, *Principles of Information Systems Analysis and Design*, Academic Press: Orlando, Florida, 1986.
25. Musa, J. D., and F. N. Woomey, "SAFEGUARD Data-Processing System: Software Project Management," Bell System Technical Journal, SAFEGUARD Supplement (1975), S245-S259.
26. Phillips, R. W., "State Change Architecture, A Protocol for Executable Process Models" in Representing and Enacting Process - Proceedings of 4th International Software Process Workshop, IEEE Press, 1988.
27. Pierce, P.A., "Software Quality Framework Issues," Volume I, II, III, SAIC, San Diego, CA.
28. Reed, B., "Process Metrics Working Notes," UES, Columbus, Ohio, July 24, 1991.
29. Ross, N., "Editorial Comments on Process Metrics," IEEE Software, Volume 23, Number 7, July 1990.
30. Royce, W., "Managing the Development of Large Software Systems: Concepts and Techniques," Proceedings, WESCON, August 1970.
31. Scacchi, W., "Managing Software Engineering Projects: A Social Analysis," IEEE Transactions on Software Engineering, Volume 10, Number 1, January 1984, pp. 49-59.
32. Selby, R. W., V. R. Basili, and F. T. Baker, "Cleanroom Software Development: An Empirical Evaluation," IEEE Transactions on Software Engineering, Volume SE-13, Number 9, September 1987.
33. Vosburgh, Curtis, Wolverton, Albert, Malec, Hoben, and Liu, "Productivity Factors in Programming Environments," Proceedings of the Seventh International Conference on Software Engineering, Wash, DC. IEEE Computer Society, 1984, pp. 143-152.
34. Walston, F., "A Method of Programming Measurement and Estimation," IBM Systems Journal, Volume 16 Number 1, 1977, pp. 54-73.
35. Weinberg, G. *The Psychology of Computer Programming*, New York: Van Nostrand Reinhold, 1971.
36. "CEPA: The Cleanroom Engineering Process Assistant," STARS Task IS-15, July 10, 1991.
37. "The Cleanroom Engineering Software Development Process," STARS Task IR-70E, CDRL Sequence 07001-001, February 28, 1991.
38. "A Software Process Management System for the STARS Software First Life Cycle," IBM STARS Deliverable CDRL Number 3016, 29 October 1990.
39. "User's Manual for SPMS," IBM STARS Deliverable CDRL Number 3118, 17 June 1991.
40. "RADC Quality Framework (Technical Report (Interim) Volume IV Software Quality Framework," Contract #F30602-88-C-0019, CDRL Sequence # A007, Software Productivity Solutions, September 1989.

Appendix A. SPMS Training Class Materials

This section includes the "SPMS Training Class" student handout.

Training Materials for the

SPMS

Software Process Management System

Click Anywhere To Begin

Another great tool from the SPI Group

Course Objectives

The course will focus on :

- Using the Software Process Management System
- Learning to create process models
- Learning how each element of the SPMS interacts with other elements
- Learning how to use the project specific plans to assist in validation of the process models

By the end of the course , students will have:

- Become familiar with the elements of the SPMS
- Created, instantiated, and simulated their own process models

Course Schedule

Day 1

9:00 Course Introduction and Overview

SPMS Overview

Process Models vs Project Plans

Granularity Issues

Nodes, Constraints, Phases

Architectural Levels

Development Modes

Hierarchical Models

Measurement Model

Graphical Representation

10:00 Building a simple Model in Xpert

Exercise 1

10:30 SPMS model representation

Exercise 2

11:00 Project Specific Data

11:30 Exercise 3

12:00 Lunch

1:00 Discussion of exercises

1:30 Scheduling the Plan

2:00 Exercise 4

2:30 Graphic Monitoring

3:00 Simulation of the Plan

3:30 Exercise 5

4:00 Discussion and review

Day 2

9:00 Review/Questions

9:30 Validation tasks and Rework

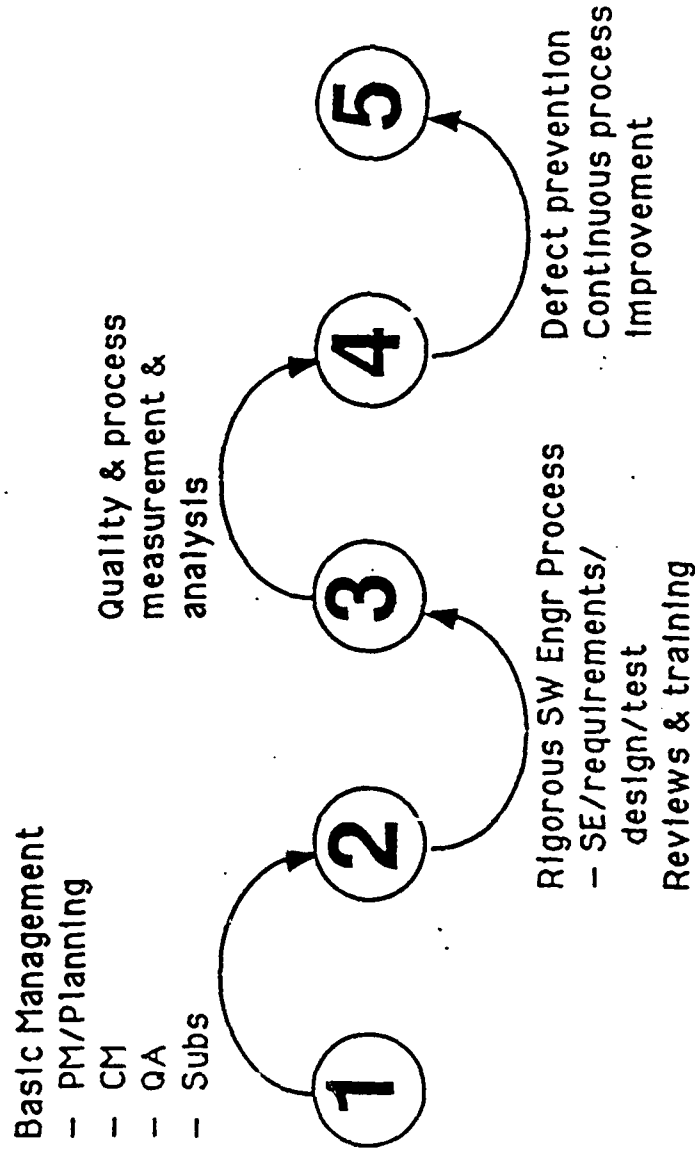
10:00 More Complex Models

10:30 Exercise 6

11:00 Discussion & Questions

11:30 Summary and support issues.

Process Capability Improvement Focus



What is Software Process Management?

Process Model = How

Project Data = What

Plan = How + What

Resources = Who

Durations + Scheduling = When

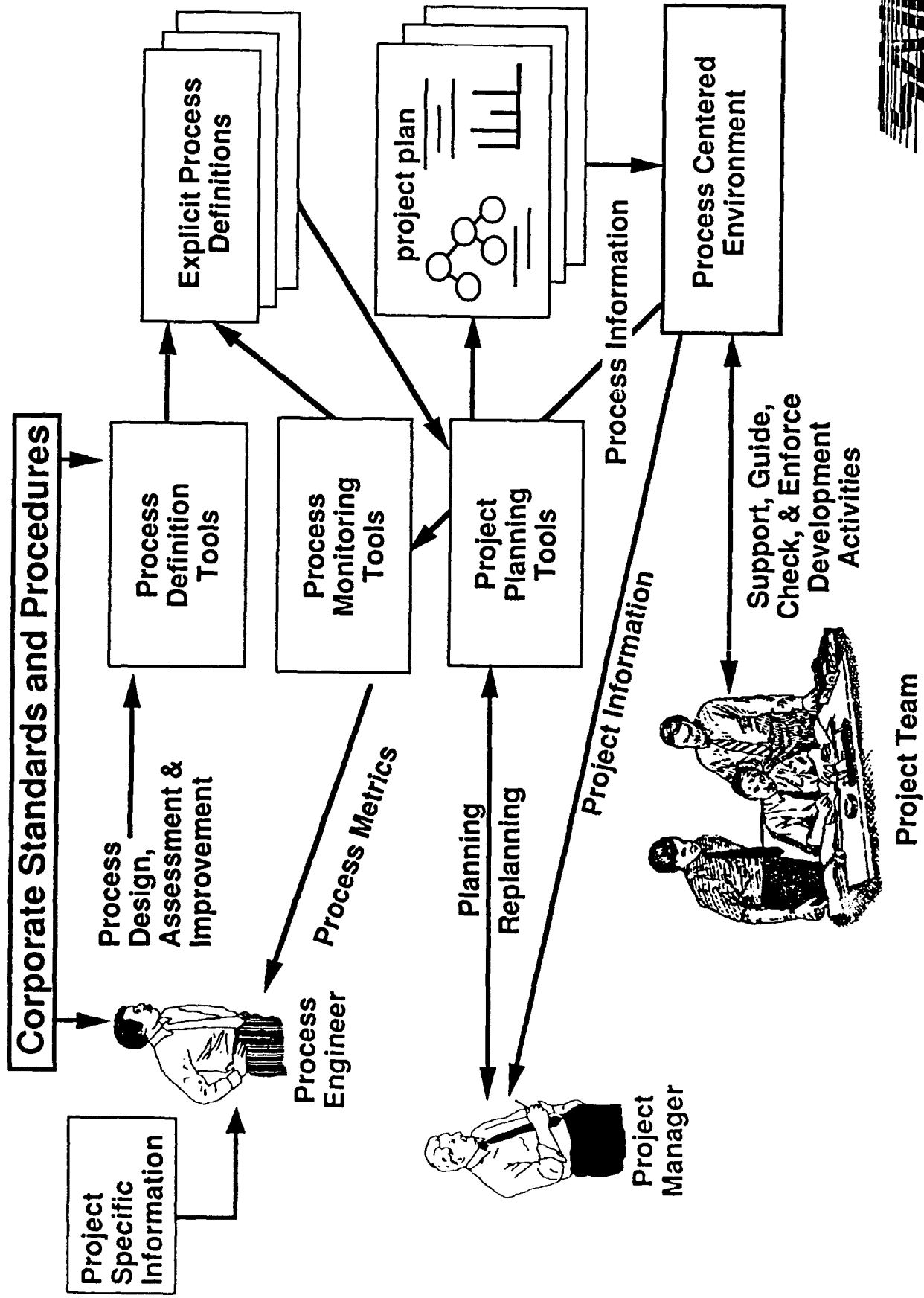
Scheduled Plan with Resources = Who, What, When, How

This Plan + monitoring methods + enaction =

Software Process Management



Process Management Paradigm



Process Models vs Instantiated Plans

Process Model:

Represents a prototypical sequence of tasks, milestones, constraints, and products necessary to produce a prototypical single instance of each of the types of software component within the process model.

Process models provide the framework for producing plans that may be replicated and a framework for metrics which may measure the process.

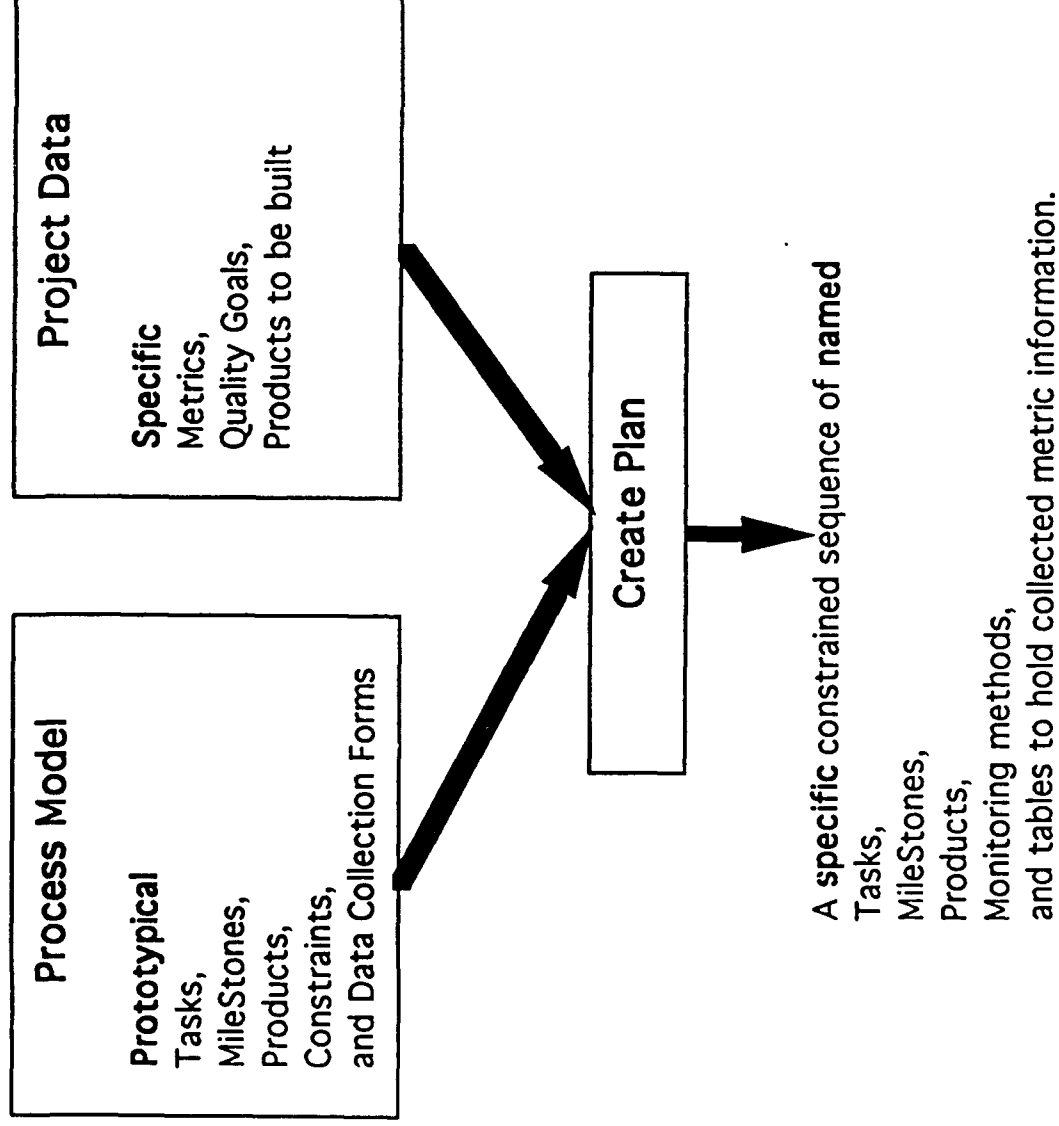
Plans:

Contain numerous specific named and inter-related instances of the software components and the tasks necessary to produce them according to the process model.

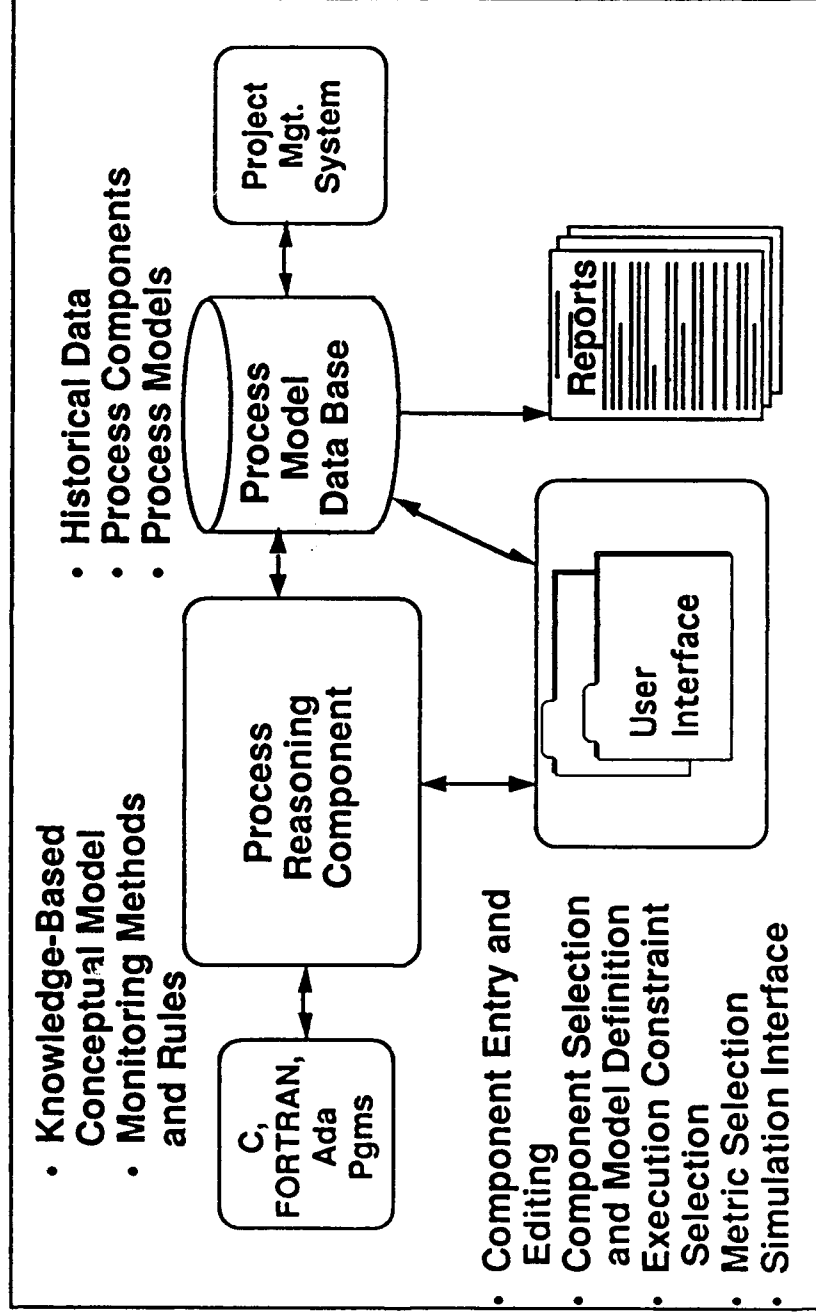
Plans are the baselines for monitoring progress of a specific software development project.



Model + Project Data = Plan



Architecture of SPMS



Process Model Granularity Issues

Process Interface



Import Graphic Model



Edit Process



Edit IO and Constraints



Edit a Model



Edit Product



Edit Constraints



Delete Model



Project Interface

- Entire model
 - Import Graphic Model
 - Delete Model
- Named Groups of Components
 - Edit Model
- Individual Components
 - Edit Process
 - Edit Product
 - Edit Constraints
 - Edit IO and Constraints

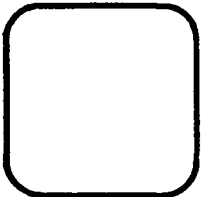


Nodes



- Task:

The basic component of a process model.



- Milestone:

A special node used to highlight important events in a process model.

- Interface:



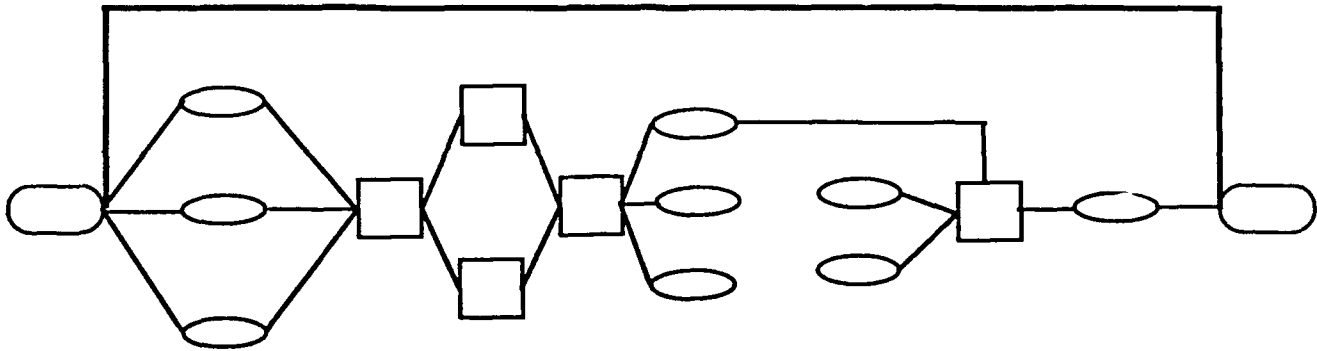
A special node used to link subnetworks in the process model. Represents a product in the SPMS.

- Reverse: (OR)



A special node used which allows the successor to start when *any* predecessor is complete rather than when *all* predecessors are complete.

Systems Analysis Phase



- Specifies that a task cannot start until its predecessor is complete.

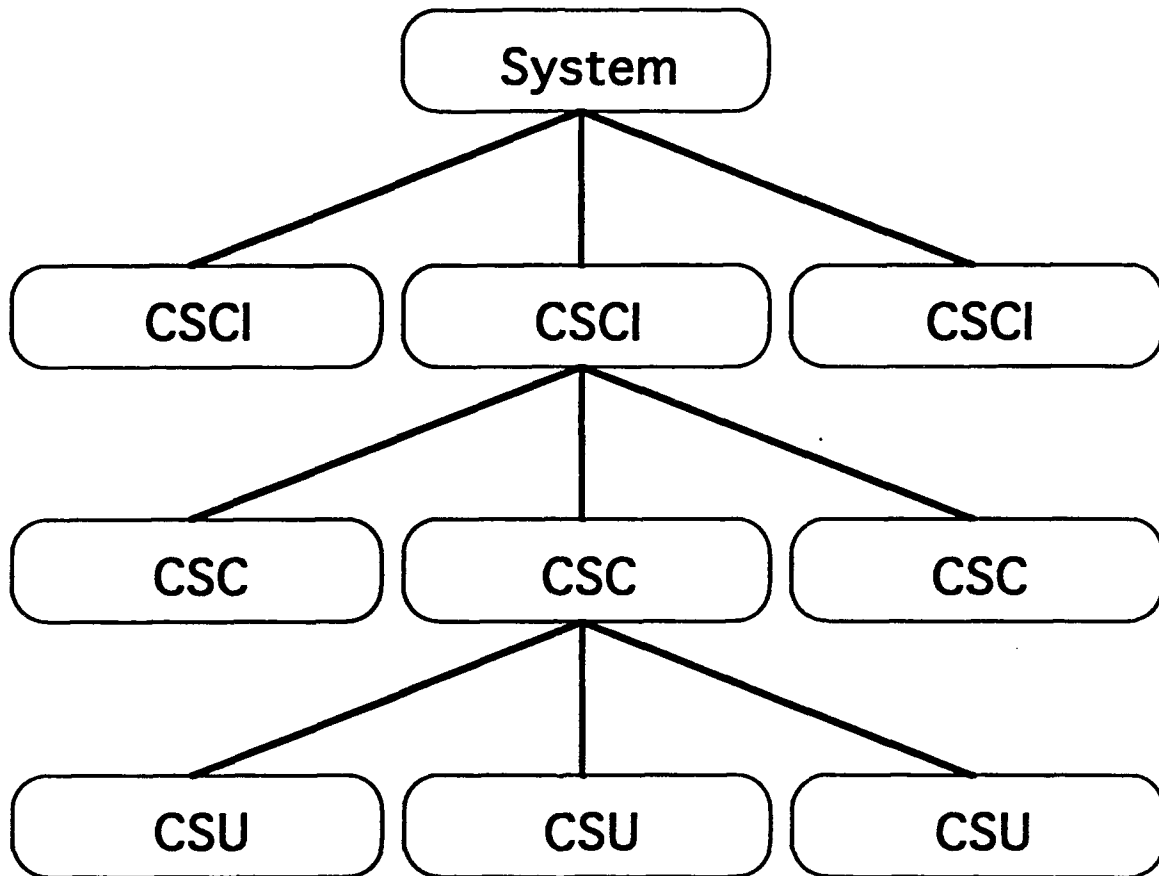
- Specifies that the completion of a task dictates the completion of its successor. May have a duration to specify lag time between a task and its successor.

- Specifies that a task cannot finish until its predecessor starts.

- Specifies that two tasks can start together. A duration applied to this type of constraint indicates a lead time between the start of one task and the start of its predecessor.

- May have resources
- Dynamic durations calculated as elapsed time between end points.
- Used to represent phases in SPMS

Architectural Levels



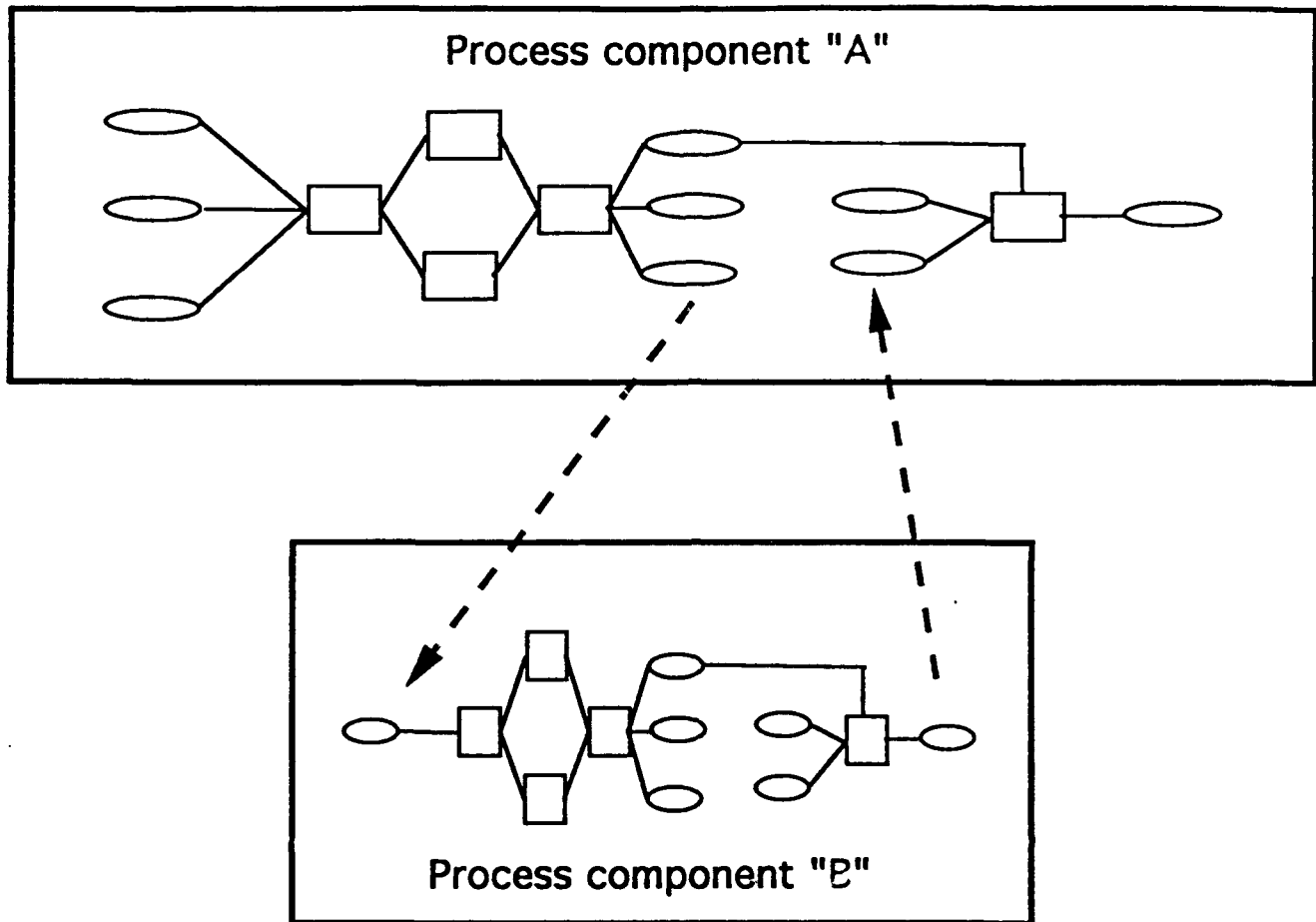
- Alternative architectural levels within a process model
- Nodes in a process model contain an architectural level parameter.
- Project specific software components also contain this parameter.
- Plans contain nodes in which the model and component parameters match on both architectural level and development mode.

Development Modes

- Develop module?
 - Reuse module?
 - Prototype module?
 - User defined....
-
- Alternative sequences of nodes within a process model
 - Nodes in a process model contain a development mode parameter.
 - Project specific software components also contain this parameter.
 - Plans contain nodes in which the model and component parameters match on both architectural level and development mode.
 - Allows instantiation time tailoring of model.



Hierarchical Models



Some Possibilities:

Process component "A" might be at the SYSTEM level.

Process component "B" might be at the CSCI level.

"B" might be considered "Part of" "A"

Process components "A" and "B" might be at the same level and "B" specify greater detail than other portions of "A". "B" might be considered "Part of" "A"

How do you want to view parts in the model editor?

Executable Process Models and Metrics

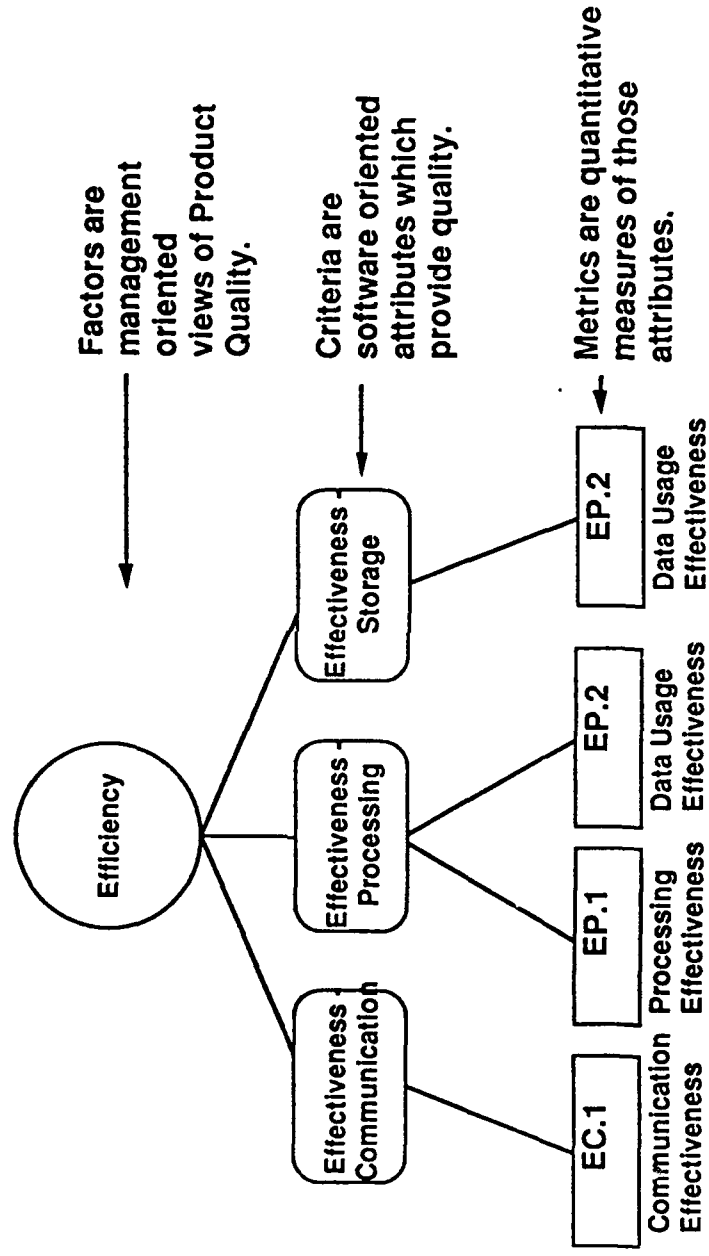
"Define the process model, then the metrics. You must know what you are doing before measuring how well you do it... The need to collect metrics on a process and items it produces... is defined in the exit criteria for that process. Thus, the process model for data collection and analysis becomes integral to the production process model ... An important secondary discovery about the interaction of metrics and process models was that metric collection could drive model refinement." (N. Ross, July'90)

Without metrics the exit criteria for executable process components cannot be evaluated

Without process models one cannot effectively identify what or when to measure, or know what actions should be taken as a result of metric evaluation

Both metrics and process models are integral to SPMS's

RADC Quality Framework



**TABLE 1.1-3 SOFTWARE QUALITY FRAMEWORK
FACTORS AND ASSOCIATED CRITERIA**

<div> <div>FACTOR</div> <div>CRITERION</div> </div>	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	F L E X I B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
ACCURACY ANOMALY MANAGEMENT AUTONOMY DISTRIBUTEDNESS EFFECTIVENESS - COMMUNICATION EFFECTIVENESS - PROCESSING EFFECTIVENESS - STORAGE OPERABILITY RECONFIGURABILITY SYSTEM ACCESSIBILITY TRAINING			X X	X X X									
COMPLETENESS CONSISTENCY TRACEABILITY VISIBILITY						X X X	X X	X X					
APPLICATION INDEPENDENCE AUGMENTABILITY COMMONALITY DOCUMENT ACCESSIBILITY FUNCTIONAL OVERLAP FUNCTIONAL SCOPE GENERALITY INDEPENDENCE SYSTEM CLARITY SYSTEM COMPATIBILITY VIRTUALITY									X X X X		X X X X		X X X X
MODULARITY SELF-DESCRIPTIVENESS SIMPLICITY			X	X			X X X	X X X	X X X	X X X	X X X	X X	X X X

AD-A255 945

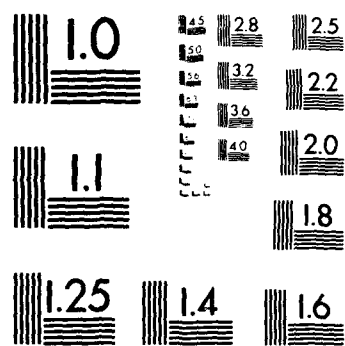
SOFTWARE TECHNOLOGY FOR ADAPTABLE RELIABLE SYSTEMS
(STARS) PROGRAM SOFTWARE (U) IBM FEDERAL SECTOR DIV
GAITHERSBURG MD W H ETT 30 SEP 91 03705-001 XC-AFSC
F19628-88-D-0032

373

UNCLASSIFIED

NL

END
FILMED
DTIC



Data Collection Forms by Phase and Architectural Level

Software Development Phase	Form
System requirements analysis/design	DCF A
Software requirements analysis	DCF B
Preliminary design	DCF C
Detailed design	DCF D
Coding and CSU testing	DCF E
CSC integration and test	DCF F
CSCI testing	DCF G
System testing	DCF H
Operational test and evaluation	DCF I

Architectural Level	Data Collection Forms								
	A	B	C	D	E	F	G	H	I
SYSTEM	X							X	X
CSCI		X	X	X	X	X	X		
CSC			X	X	X	X			
CSU				X	X	X			





Component Boo, 1.1.1

DCF A-Level

AC.1.1.1.a

Is there a reference to available documentation which describes the results of an error analysis?

☐ Yes

☐ No

☐ Not Applicable



Contents

DCF A-Level

AC.1.2.a

Have accuracy requirements been budgeted to the individual capabilities?

☐ Yes

☐ No

☐ Not Applicable



More...

B

C

D

E

F

G

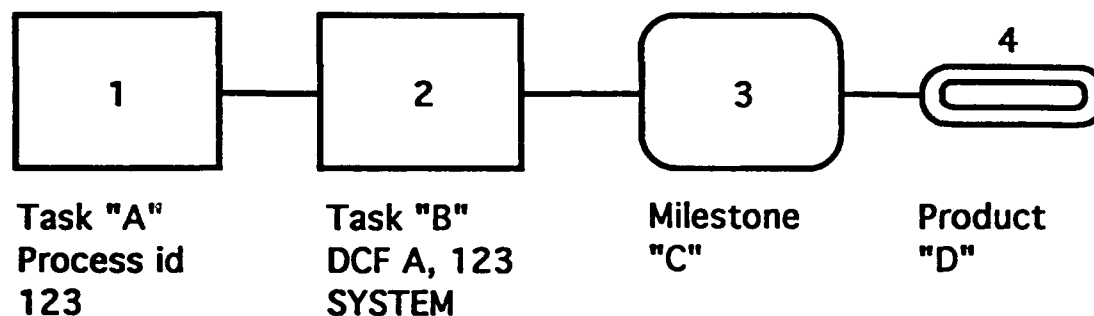
H

I

**Appendix A. Metric Scoring Formulas Based on Metric Elements
by Architecture Level and Phase**

METRIC	DCF	LEVEL	METRIC ELEMENTS FORMULA
	b	CSCI	(SI.2.1.b)
	c	CSCI	(SI.2.1.c)
	d	CSU	(SI.2.1.d)
	g	CSCI	(SI.2.1.g)
	h	System	(SI.2.1.h)
SI.3	d	CSU	$1 / (1 + SI.3.1.d + SI.3.2.d)$
	e	CSU	$1 / (1 + SI.3.1.e + SI.3.2.e)$
SI.4	d	CSU	$a * (SI.4.1.d) + b * (1 - (SI.4.3.d / AU.1.2.d)) + c * (1 - (SI.4.5.d / SI.4.6.d)) + d * (1 - (SI.4.7.d / SI.4.6.d)) + e * (SI.4.8.d) + f * (1 / SI.4.10.d) + g * (1 - (SI.4.11.d / AU.1.2.d)) + h * (1 - (SI.4.12.d + SI.4.13.d) / AU.1.2.d) + i * (SI.4.15.d / SI.4.14.d) + j * (SI.4.16.d)$
	d	CSCI	(SI.4.18.d)
	e	CSU	$a * (SI.4.1.e) + b * (SI.4.2.e) + c * (1 / (1 + SI.4.3.e)) + d * (1 - (SI.4.5.e / SI.4.6.e)) + e * (1 - (SI.4.7.e / SI.4.6.e)) + f * (SI.4.8.e) + g * (1 - (SI.4.9.e / AP.3.3.e)) + h * (1 / SI.4.10.e) + i * (1 - (SI.4.11.e / AP.3.3.e)) + j * (1 - (SI.4.12.e + SI.4.13.e) / AP.3.3.e) + k * (SI.4.15.e / SI.4.14.e) + l * (SI.4.16.e) + m * (SI.4.17.e)$
	e	CSCI	(SI.4.18.e)
SI.5	d	CSU	$a * (1 / (1 + SI.5.1.d)) + b * (SI.5.3.d / SI.5.2.d) + c * (SI.5.4.d)$
	e	CSU	$a * (1 / (1 + SI.5.1.e)) + b * (SI.5.3.e / SI.5.2.e) + c * (SI.5.4.e)$

Validation Tasks and Rework Node Id'S

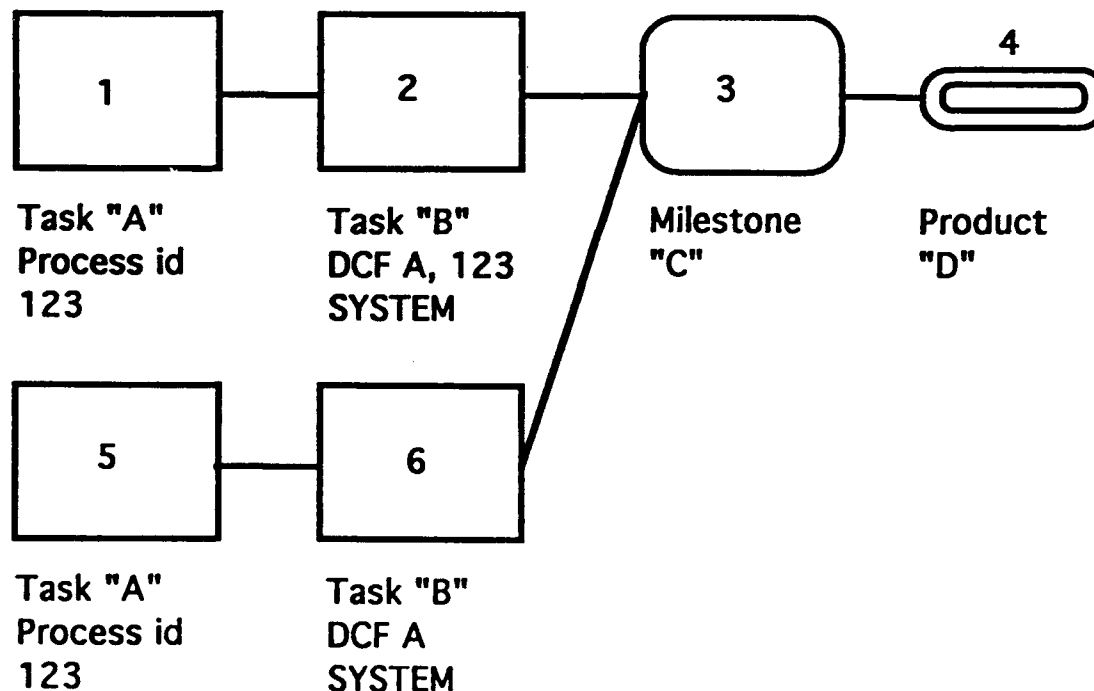


Task "B" is a validation task.

When Task "B" begins it requests that the Data Collection Form (DCF A) associated with it be filled with data.

When the data becomes available, the metrics associated with this project are computed and compared with the quality goals of Product "D".

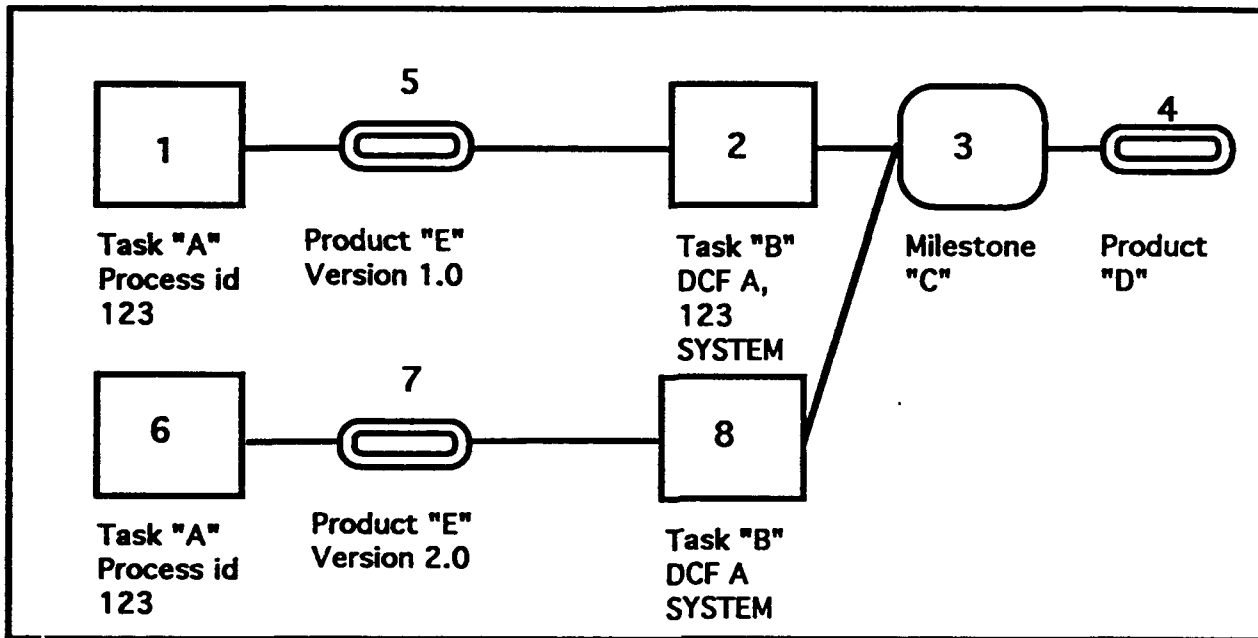
- Quality goals met, then Task "B" is complete and the Milestone "C" is met and Product "D" becomes available.
- Quality goals not met then replanning will find the task associated with product "D" which is of the process type "123" and use it as the starting point for cloning up to and including the validation task.



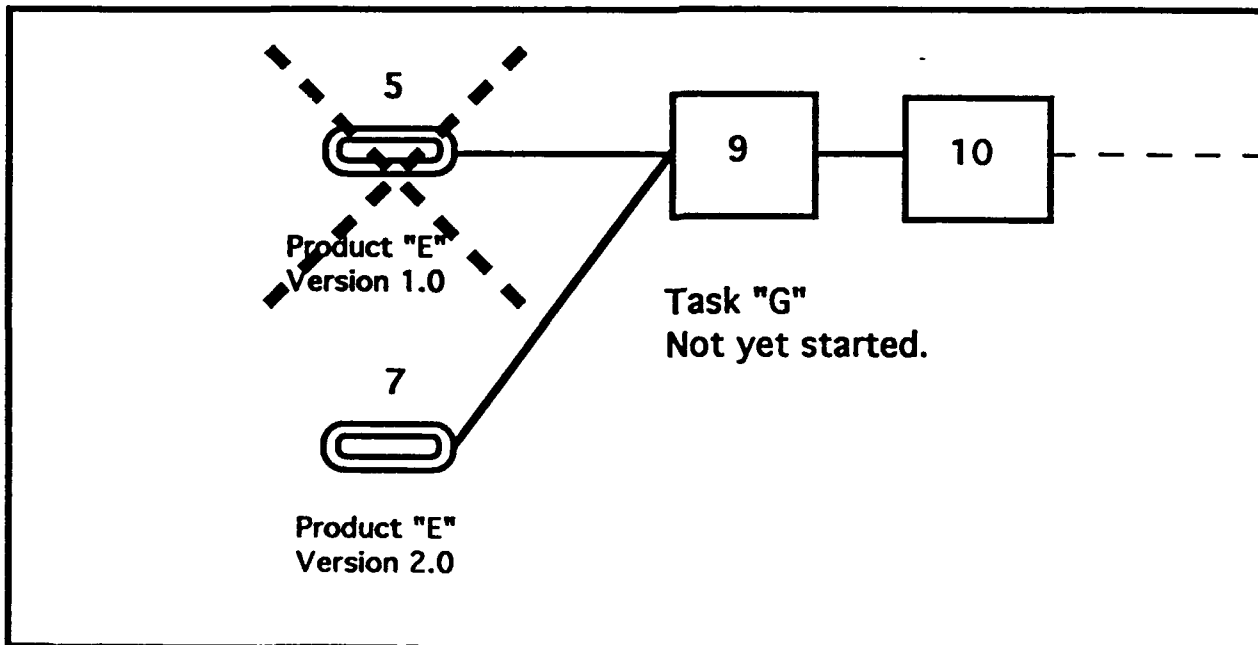
Impact of Re-Work on other Process Components

Tasks Not Started Case

Process Component 1

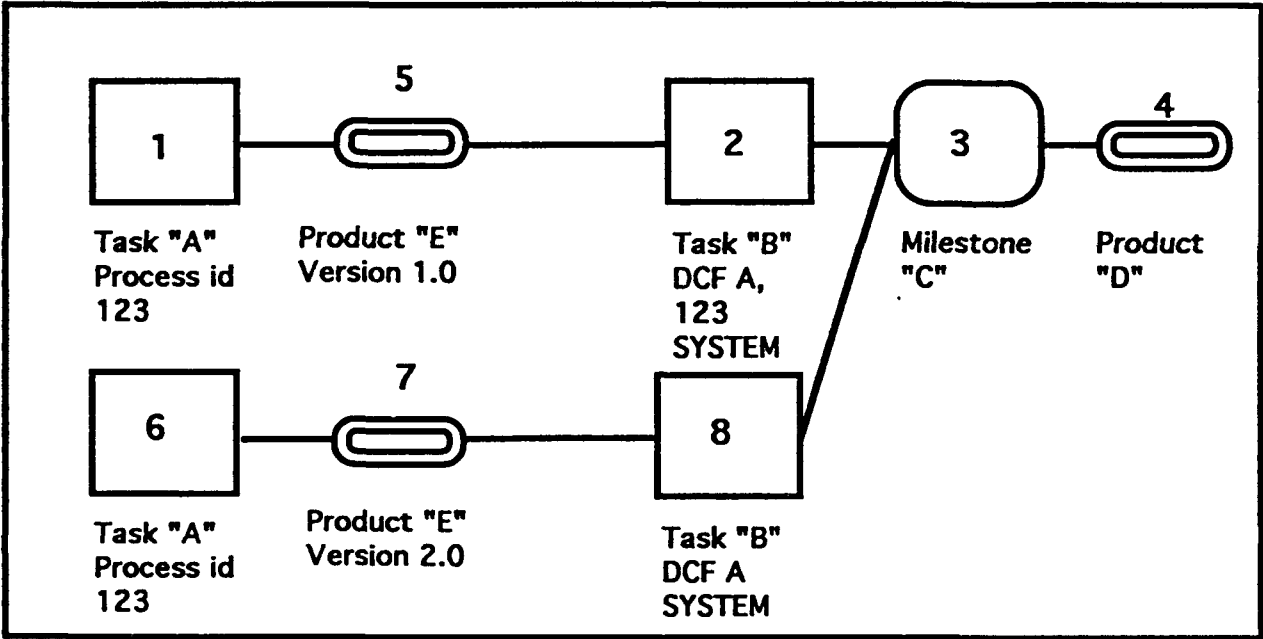


Process Component 2

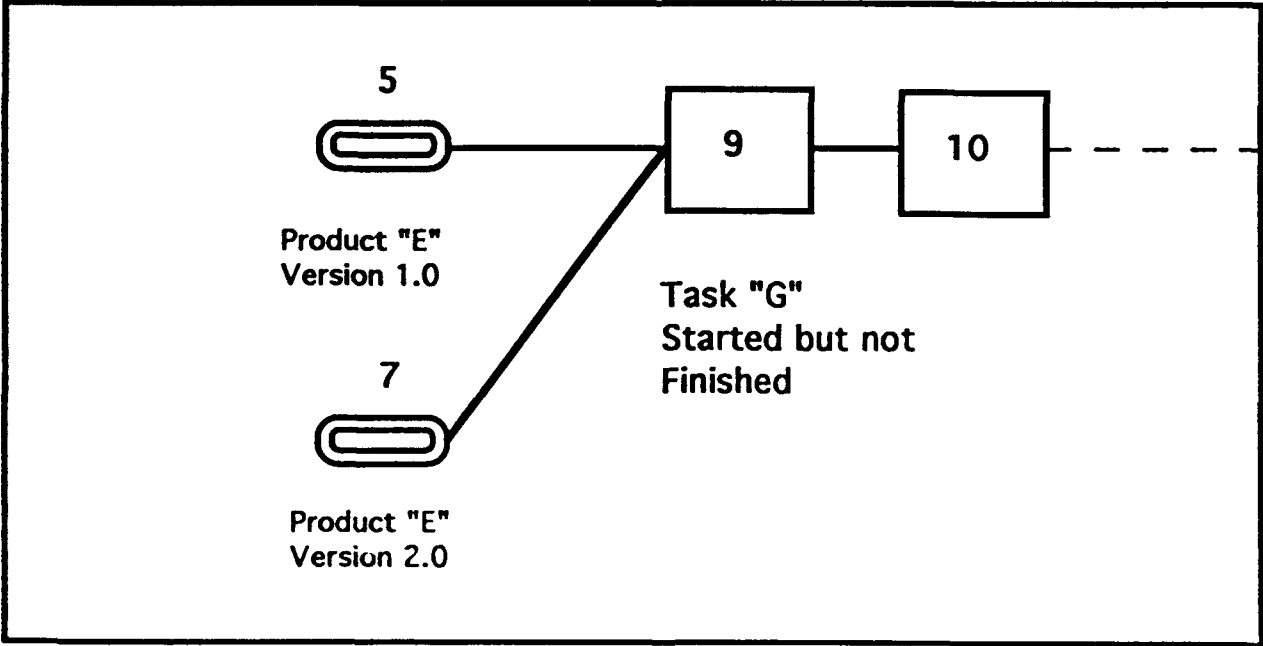


Impact of Re-Work on other Process Components Tasks In Progress Case

Process Component 1



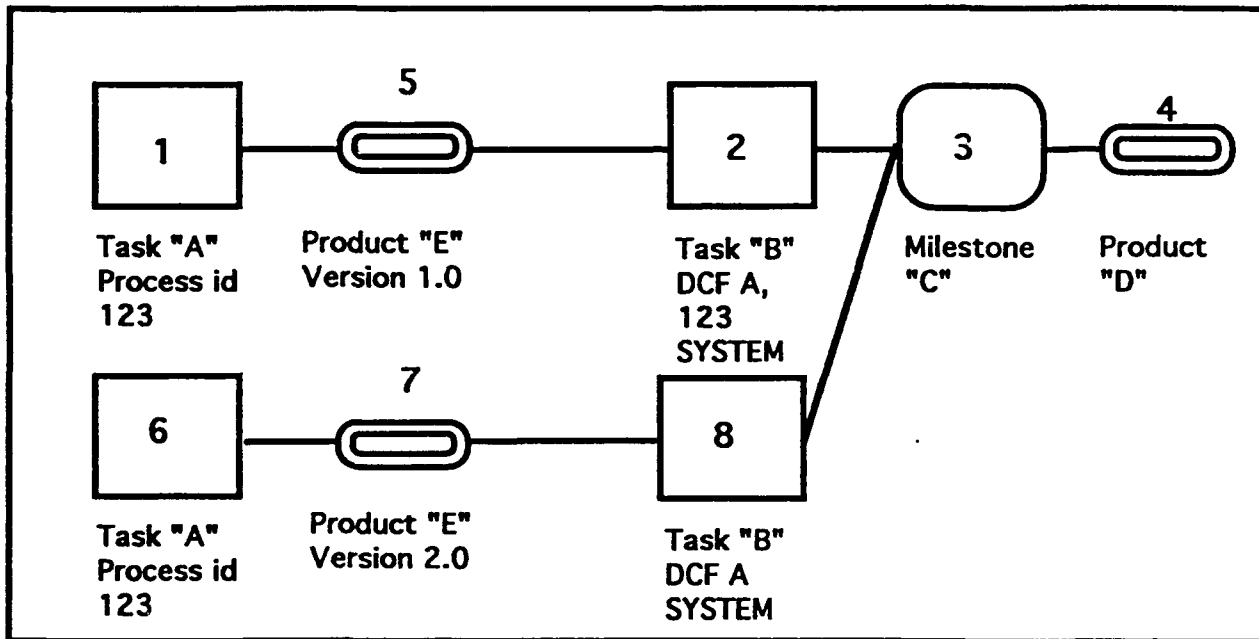
Process Component 2



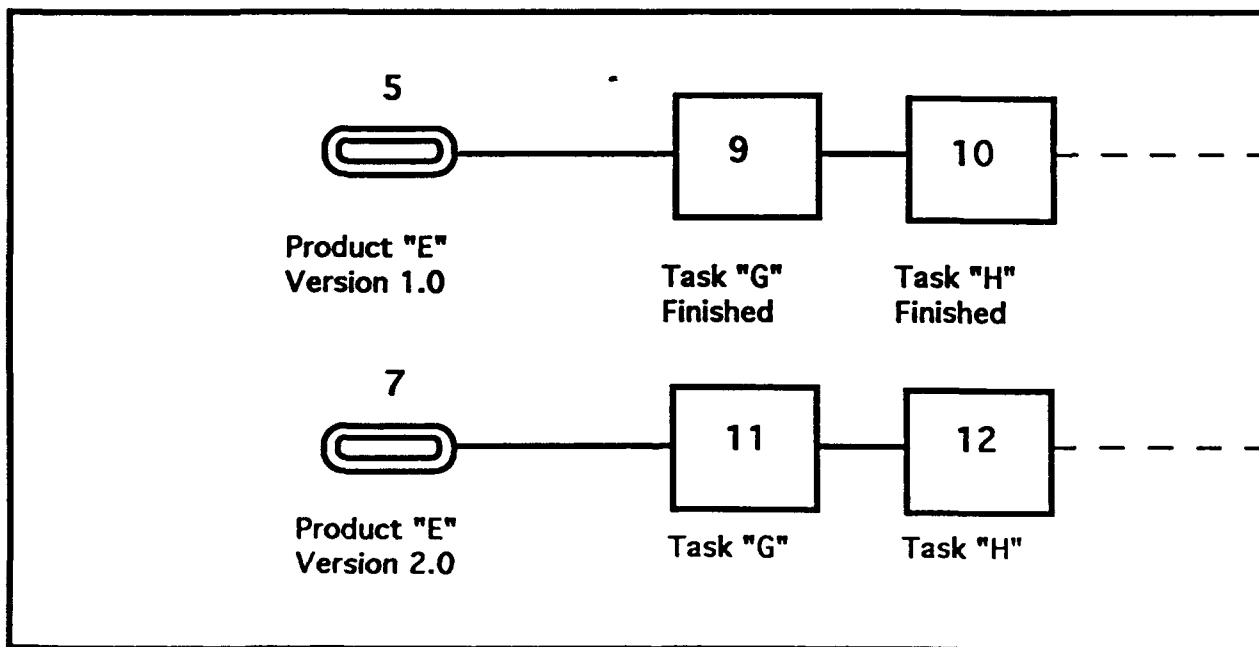
Impact of Re-Work on other Process Components

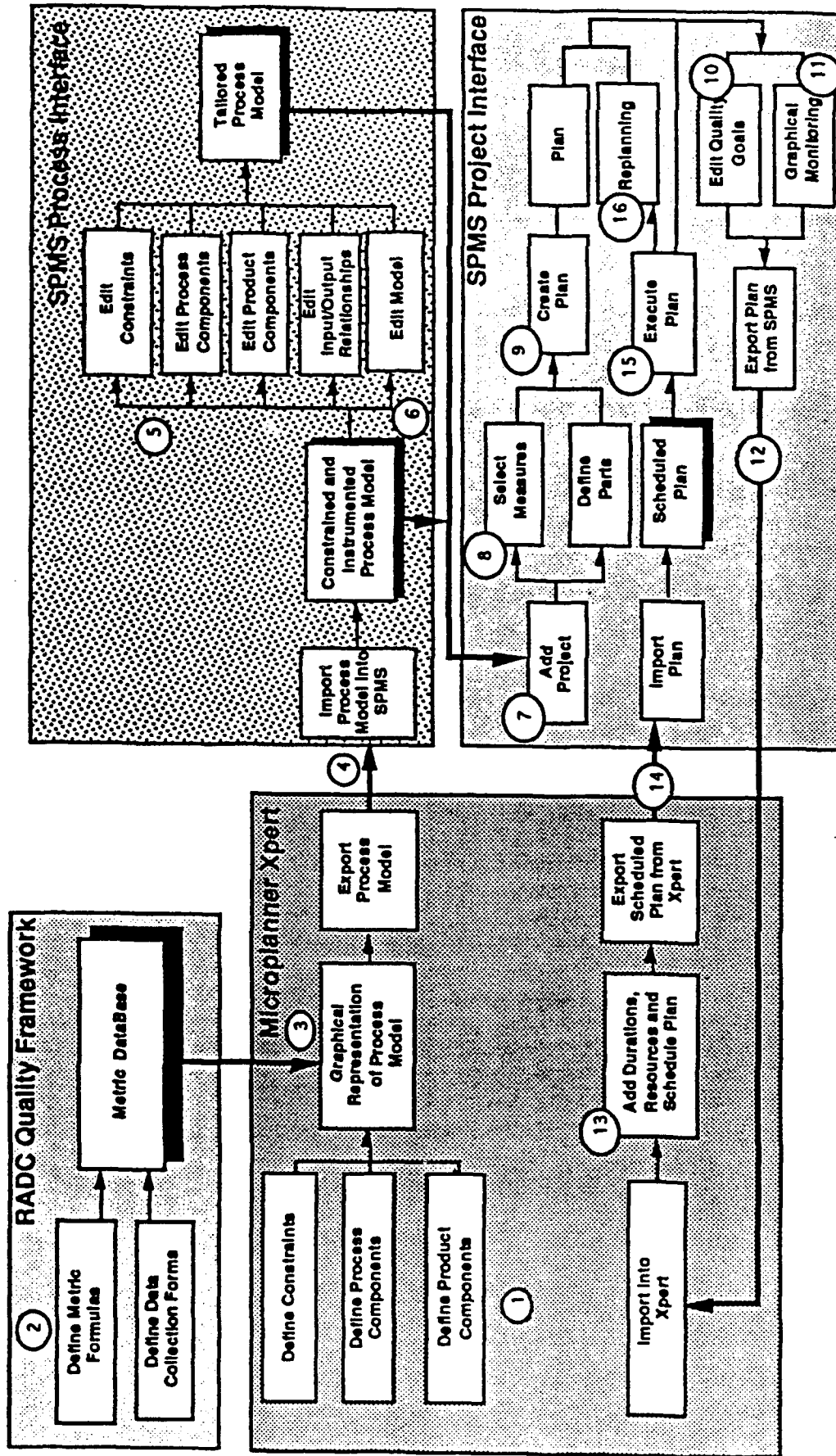
Tasks Complete Case

Process Component 1



Process Component 2





Exercise 1

1. Using the associated hand-out input the System Requirements Analysis subset of the process model as a part of a larger model.
2. Create a Systems Analysis Phase .
3. Use only Finish-to-Start constraints.
4. Export the model from Xpert.

6. DEVELOPMENT PROCESS

6.0.1 The developer shall implement a process for developing the software. The development process will be composed of the following major activities:

- a. System Requirements Analysis
- b. System Design
- c. Software Requirements Analysis
- d. Software Architectural Design
- e. Software Detailed Design
- f. Software Coding and Testing
- g. Software Integration and Testing
- h. Software Configuration Item Testing
- i. System Integration and Testing
- j. System Testing
- k. System Installation and Checkout.

6.0.2 The developer shall select and map these activities onto the life cycle model established for the software project. The selected activities may overlap and may be performed iteratively or recursively.

6.0.3 The developer shall use the methodologies, standards, and procedures that are systematic, adequately documented, and established by the developer's organization for performing the activities.

6.0.4 The developer shall use the computer programming language(s) as specified in the contract to code the deliverable software.

6.0.5 The developer shall consider incorporating non-developmental items into the deliverable software. Incorporation of such items shall comply with the documentation, data rights, warranty, and other requirements as specified in the contract.

6.0.6 The developer may employ non-deliverable items in the development or maintenance of software. However, the developer shall ensure that the operation and maintenance of software after its delivery to the purchaser will be independent of such non-deliverable items. In case such independence cannot be ensured, the developer shall treat the non-deliverable items as non-developmental upon notifying the purchaser regarding the impact of non-deliverable items on the cost, schedule, operation, and maintenance of the deliverable software.

6.1 System Requirements Analysis. The developer shall perform or support the following system requirements analysis activities.

6.1.1 Engineering. The developer shall analyze the statement of need, statement of work, and recommended solutions, if available, to define a set of system requirements addressing the following as a minimum:

- a. Functions and capabilities of the total system, including performance, quality, and physical characteristics and environmental conditions under which the system will perform;
- b. Safety requirements, including those related to equipment characteristics and degradation, methods of operation and maintenance, environmental influences, and personnel injury;
- c. Security requirements, including those related to operational and maintenance environments and compromise of sensitive information or materials;
- d. Human engineering requirements, including those related to constraints on personnel, areas needing concentrated human attention and sensitive to human errors, and training;
- e. Interfaces requirements for interfaces external to the system, including interfaces with users;
- f. Operation and maintenance requirements.

6.1.2 Qualification Testing. The developer shall define a set of system qualification requirements, including qualifications methods, for verification, validation, and testing of the system requirements.

6.1.3 Documentation. The developer shall document the system and qualification requirements in a system requirements document in accordance with section 9.4.

6.1.4 Product Evaluation. The developer shall perform evaluations of the system and qualification requirements for the criteria identified below as a minimum. When a problem is detected, corrective actions shall be taken in accordance with section 9.3.3.

- a. Consistency -- external and internal;
- b. Traceability;
- c. Test coverage of requirements;
- d. Testability;
- e. Feasibility of design, operation, and maintenance;

6.1.5 Formal Review. The developer shall conduct one or more system requirements reviews in accordance with section 9.2.

6.1.6 Configuration Management. The developer shall place the documents identified below under configuration management and perform configuration control in accordance with section 9.1:

- a. Statement of work
- b. System requirements document.

Exercise 2

1. Import the graphic model of Exercise 1 into the SPMS.
2. Browse the fine grained components using "Edit Process", "Edit Products", and "Edit Constraints" buttons.
3. Browse the product to producer and consumer relations using the "Edit IO and Constraints" button
4. Browse the hierarchical coarse grained components using the "Edit Model" button
5. Edit your model as desired .

Exercise 3

1. Move to the Project Interface and create a new Project.
2. Define a system level component with a source to match the development mode of your model.
3. Create a Plan by selecting your model and your project and providing a name for your plan.
4. Export the plan from the SPMS

Exercise 4

1. Create a new project in XPERT. Clear the New Subproject which is automatically created. From the Date Control Panel, Turn off the Show Hours/Minutes option .
2. Import your plan into XPERT.
3. Use "Clean Up" to improve the readability of the activity network.
4. Select the tasks in the network and enter durations. (Format is "weeks,days").
5. Perform Time Analysis.
6. Build a Gannt chart to display your results. See Gannt chart options for tailoring the chart.
7. Build a Table view. Use "Selected Table" with "0" resources displayed
8. Select the entire table and Export it from XPERT. (use name.DAT" format)

Exercise 5

1. Import the scheduled plan into the SPMS.
2. Randomize the durations of your plan.
3. Create some graphs for monitoring your plan during execution
4. Execute the plan.

Exercise 6

1. Open your model in Xpert.
2. Add a validation task for the Systems Requirements Document.
3. Be sure your DCF and rework node id are appropriate!
4. Alter the constraints to allow some tasks to begin when *any* of their inputs are available but only finish when *all* of their required inputs have been made available.
5. Export the model from Xpert.
6. Import the model into SPMS
7. Create a new project.
8. Select measures and default quality goals for this project.
9. Define Parts.
10. Create a new plan using your new model and new project.
11. Execute the plan.
12. Replan as necessary.